Data Processing, Analysis and Visualization in Python

Basic Machine Learning III – Supervised Learning

1

Outline

- Introduction
- Data splitting
- Data standardization
- Performance metrics (confusion matrix, recall, precision, F1 score)
- Regression
  - Linear regression
  - Multiple linear regression
  - LASSO
- Logistic regression
- Decision trees
- k-Nearest Neighbors

2

Supervised Learning in Scikit-learn

- Linear Models
- Linear and Quadratic Discriminant Analysis
- Kernel ridge regression
- Support Vector Machines
- Stochastic Gradient Descent
- Nearest Neighbors
- Gaussian Processes
- Cross decomposition
- Naive Bayes
- Decision Trees
- Ensemble methods
- Multiclass and multioutput algorithms
- Feature selection
- Semi-supervised learning
- Isotonic regression
- Probability calibration
- Neural network models (supervised)

3

Supervised Learning in Scikit-learn

- Linear Models
  - Ordinary Least Squares
  - Ridge regression and classification
  - Lasso
  - Multi-task Lasso
  - Elastic-Net
  - Multi-task Elastic-Net
  - Least Angle Regression
  - LARS Lasso
  - Orthogonal Matching Pursuit (OMP)
  - Bayesian Regression
  - Logistic regression
  - Generalized Linear Regression
  - Stochastic Gradient Descent - SGD
  - Perceptron
  - Passive Aggressive Algorithms
  - Robustness regression: outliers and modeling errors
  - Quantile Regression
  - Polynomial regression: extending linear models with basis functions
- Linear and Quadratic Discriminant Analysis
- Kernel ridge regression
- Support Vector Machines
- Stochastic Gradient Descent
- Nearest Neighbors
- Gaussian Processes
- Cross decomposition
- Naive Bayes
- Decision Trees
- Ensemble methods
- Multiclass and multioutput algorithms
- Feature selection
- Semi-supervised learning
- Isotonic regression
- Probability calibration
- Neural network models (supervised)

4

Supervised Learning in Scikit-learn

- Linear Models
- Linear and Quadratic Discriminant Analysis
  - Dimensionality reduction using Linear Discriminant Analysis
  - Mathematical formulation of the LDA and QDA classifiers
  - Mathematical formulation of LDA dimensionality reduction
  - Shrinkage and Covariance Estimator
  - Estimation algorithms
- Kernel ridge regression
- Support Vector Machines
- Stochastic Gradient Descent
- Nearest Neighbors
- Gaussian Processes
- Cross decomposition
- Naive Bayes
- Decision Trees
- Ensemble methods
- Multiclass and multioutput algorithms
- Feature selection
- Semi-supervised learning
- Isotonic regression
- Probability calibration
- Neural network models (supervised)

5

Supervised Learning in Scikit-learn

- Linear Models
- Linear and Quadratic Discriminant Analysis
- Kernel ridge regression
- Support Vector Machines
  - Classification
  - Regression
  - Density estimation, novelty detection
  - Implementation details
- Stochastic Gradient Descent
- Nearest Neighbors
- Gaussian Processes
- Cross decomposition
- Naive Bayes
- Decision Trees
- Ensemble methods
- Multiclass and multioutput algorithms
- Feature selection
- Semi-supervised learning
- Isotonic regression
- Probability calibration
- Neural network models (supervised)

6

Supervised Learning in Scikit-learn

- Linear Models
- Linear and Quadratic Discriminant Analysis
- Kernel ridge regression
- Support Vector Machines
- Stochastic Gradient Descent
  - Classification
  - Regression
  - Online One-Class SVM
  - Stochastic Gradient Descent for sparse data
- Nearest Neighbors
- Gaussian Processes
- Cross decomposition
- Naive Bayes
- Decision Trees
- Ensemble methods
- Multiclass and multioutput algorithms
- Feature selection
- Semi-supervised learning
- Isotonic regression
- Probability calibration
- Neural network models (supervised)

7

Supervised Learning in Scikit-learn

- Linear Models
- Linear and Quadratic Discriminant Analysis
- Kernel ridge regression
- Support Vector Machines
- Stochastic Gradient Descent
- Nearest Neighbors
  - Unsupervised Nearest Neighbors
  - Nearest Neighbors Classification
  - Nearest Neighbors Regression
  - Nearest Neighbor Algorithms
  - Nearest Centroid Classifier
  - Nearest Neighbors Transformer
  - Neighborhood Components Analysis
- Gaussian Processes
- Cross decomposition
- Naive Bayes
- Decision Trees
- Ensemble methods
- Multiclass and multioutput algorithms
- Feature selection
- Semi-supervised learning
- Isotonic regression
- Probability calibration
- Neural network models (supervised)

8

Supervised Learning in Scikit-learn

- Linear Models
- Linear and Quadratic Discriminant Analysis
- Kernel ridge regression
- Support Vector Machines
- Stochastic Gradient Descent
- Nearest Neighbors
- Gaussian Processes
  - Gaussian Process Regression (GPR)
  - Gaussian Process Classification (GPC)
  - Kernels for Gaussian Processes
- Cross decomposition
- Naive Bayes
- Decision Trees
- Ensemble methods
- Multiclass and multioutput algorithms
- Feature selection
- Semi-supervised learning
- Isotonic regression
- Probability calibration
- Neural network models (supervised)

9

Supervised Learning in Scikit-learn

- Linear Models
- Linear and Quadratic Discriminant Analysis
- Kernel ridge regression
- Support Vector Machines
- Stochastic Gradient Descent
- Nearest Neighbors
- Gaussian Processes
- Cross decomposition
  - PLSCanonical
  - PLSSVD
  - PLSRegression
  - Canonical Correlation Analysis
- Naive Bayes
- Decision Trees
- Ensemble methods
- Multiclass and multioutput algorithms
- Feature selection
- Semi-supervised learning
- Isotonic regression
- Probability calibration
- Neural network models (supervised)

10

Supervised Learning in Scikit-learn

- Linear Models
- Linear and Quadratic Discriminant Analysis
- Kernel ridge regression
- Support Vector Machines
- Stochastic Gradient Descent
- Nearest Neighbors
- Gaussian Processes
- Cross decomposition
- Naive Bayes
  - Gaussian Naive Bayes
  - Multinomial Naive Bayes
  - Complement Naive Bayes
  - Bernoulli Naive Bayes
  - Categorical Naive Bayes
  - Out-of-core naive Bayes model fitting
- Decision Trees
- Ensemble methods
- Multiclass and multioutput algorithms
- Feature selection
- Semi-supervised learning
- Isotonic regression
- Probability calibration
- Neural network models (supervised)

11

Supervised Learning in Scikit-learn

- Linear Models
- Linear and Quadratic Discriminant Analysis
- Kernel ridge regression
- Support Vector Machines
- Stochastic Gradient Descent
- Nearest Neighbors
- Gaussian Processes
- Cross decomposition
- Naive Bayes
- Decision Trees
  - Classification
  - Regression
  - Multi-output problems
  - Complexity
  - Tips on practical use
  - Tree algorithms: ID3, C4.5, C5.0 and CART
  - Mathematical formulation
  - Minimal Cost-Complexity Pruning
- Ensemble methods
- Multiclass and multioutput algorithms
- Feature selection
- Semi-supervised learning
- Isotonic regression
- Probability calibration
- Neural network models (supervised)

12

Supervised Learning in Scikit-learn

- Linear Models
- Linear and Quadratic Discriminant Analysis
- Kernel ridge regression
- Support Vector Machines
- Stochastic Gradient Descent
- Nearest Neighbors
- Gaussian Processes
- Cross decomposition
- Naive Bayes
- Decision Trees
- Ensemble methods
  - Bagging meta-estimator
  - Forests of randomized trees
  - AdaBoost
  - Gradient Tree Boosting
  - Histogram-Based Gradient Boosting
  - Voting Classifier
  - Voting Regressor
  - Stacked generalization
- Multiclass and multioutput algorithms
- Feature selection
- Semi-supervised learning
- Isotonic regression
- Probability calibration
- Neural network models (supervised)

13

Supervised Learning in Scikit-learn

- Linear Models
- Linear and Quadratic Discriminant Analysis
- Kernel ridge regression
- Support Vector Machines
- Stochastic Gradient Descent
- Nearest Neighbors
- Gaussian Processes
- Cross decomposition
- Naive Bayes
- Decision Trees
- Ensemble methods
- Multiclass and multioutput algorithms
  - Multiclass classification
  - Multilabel classification
  - Multiclass-multioutput classification
  - Multioutput regression
- Feature selection
- Semi-supervised learning
- Isotonic regression
- Probability calibration
- Neural network models (supervised)

14

Supervised Learning in Scikit-learn

- Linear Models
- Linear and Quadratic Discriminant Analysis
- Kernel ridge regression
- Support Vector Machines
- Stochastic Gradient Descent
- Nearest Neighbors
- Gaussian Processes
- Cross decomposition
- Naive Bayes
- Decision Trees
- Ensemble methods
- Multiclass and multioutput algorithms
- Feature selection
  - Removing features with low variance
  - Univariate feature selection
  - Recursive feature elimination
  - Feature selection using SelectFromModel
  - Sequential Feature Selection
  - Feature selection as part of a pipeline
- Semi-supervised learning
- Isotonic regression
- Probability calibration
- Neural network models (supervised)

15

Supervised Learning in Scikit-learn

- Linear Models
- Linear and Quadratic Discriminant Analysis
- Kernel ridge regression
- Support Vector Machines
- Stochastic Gradient Descent
- Nearest Neighbors
- Gaussian Processes
- Cross decomposition
- Naive Bayes
- Decision Trees
- Ensemble methods
- Multiclass and multioutput algorithms
- Feature selection
- Semi-supervised learning
  - Self Training
  - Label Propagation
- Isotonic regression
- Probability calibration
- Neural network models (supervised)

16

Supervised Learning in Scikit-learn

- Linear Models
- Linear and Quadratic Discriminant Analysis
- Kernel ridge regression
- Support Vector Machines
- Stochastic Gradient Descent
- Nearest Neighbors
- Gaussian Processes
- Cross decomposition
- Naive Bayes
- Decision Trees
- Ensemble methods
- Multiclass and multioutput algorithms
- Feature selection
- Semi-supervised learning
- Isotonic regression
- Probability calibration
  - Calibration curves
  - Calibrating a classifier
- Neural network models (supervised)

17

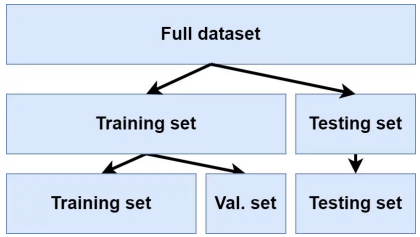
Supervised Learning in Scikit-learn

- Linear Models
- Linear and Quadratic Discriminant Analysis
- Kernel ridge regression
- Support Vector Machines
- Stochastic Gradient Descent
- Nearest Neighbors
- Gaussian Processes
- Cross decomposition
- Naive Bayes
- Decision Trees
- Ensemble methods
- Multiclass and multioutput algorithms
- Feature selection
- Semi-supervised learning
- Isotonic regression
- Probability calibration
- Neural network models (supervised)
  - Multi-layer Perceptron
  - Classification
  - Regression
  - Regularization
  - Algorithms

18

Data Splitting

- The data set should be split into training and test sets



19

3 Types of datasets – Train/Dev/Test

- Training set — Which you run your learning algorithm on
- Dev/Val (development) set — Which you use to tune parameters, select features, and make other decisions regarding the learning algorithm. Sometimes also called the *hold-out cross validation set*
- Test set — which you use to evaluate the performance of the algorithm, but not to make any decisions regarding what learning algorithm or parameters to use

20

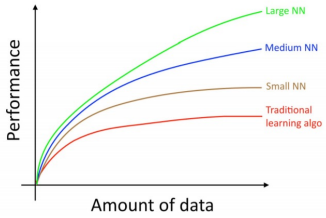
The size of the test set

- Small datasets add a lot of noise and can only detect large changes in performance
- If you care about small improvements to your algorithm – you need a large test dataset (e.g., online advertising, gaming)
- The important point is not the % of data split between train/dev/test, but instead the absolute number of dev/test samples, the representation of minority classes there, and our confidence in results obtained on a given size of the dev/test sets.

21

ML Models need data!

- If a Genie grants you 3 wishes, ask for Data, Data and more Data
- But… data should be high quality
- This is a major challenge



22

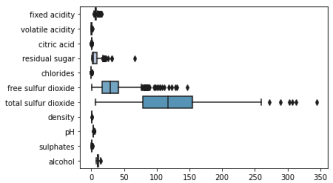
Splitting Data with SKLearn

- In general, the data is split into two groups before generating a model. These groups are called training and testing sets.
  - The training set is used for developing models and select the parameters that are adequate for the data.
  - The test set is used to assess the performance of the model with previously unseen data.
- The most common way is to divide the data is random splitting.
- We will split the dataset using the function `train_test_split` from scikit learn. The `random_state` parameter controls the shuffling applied to the data before applying the split.
- You can split the training set into training and dev.

23

Splitting Data with SKLearn

```
>>> from sklearn.model_selection import train_test_split

>>> X = df.drop(['quality', 'hue'], axis=1)
>>> y = df['hue']
>>> X_train, X_test, y_train, y_test =
          train_test_split(X, y, test_size=0.33,
          random_state=42)
>>> ax = sns.boxplot(data=X_train, orient='h',
         palette='PuBuGn')
```



regression.ipynb

24

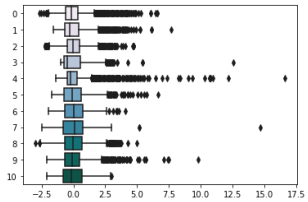
Data Standardization

- Standardization is a method to transform variables that differ in mean and deviation into comparable values.
- This process consists of **subtracting the means** from each feature and then **dividing by the feature standard deviation**.
- Many machine learning algorithms assume that all features are centered around zero and have approximately the same variance; if so, then standardization is needed.
- Here, we will use the function `StandardScaler`. It will first fit the data to determine the mean and standard deviation and then transform it into a standardized form.
- There are other processes of standardization.

25

Splitting Data with SKLearn

```
>>> from sklearn import preprocessing

>>> scaler = preprocessing.StandardScaler()
>>> scaler.fit(X_train)
>>> X_train = scaler.transform(X_train)
>>> X_test = scaler.transform(X_test)
>>> ax = sns.boxplot(data=X_train, orient='h',
        palette='PuBuGn')
```



26

Correlation

- To fit the data with a linear model regression, it is good practice to employ variables presenting a high correlation with the target. One way to do this is by calculating correlation coefficients and another way is through visual methods.

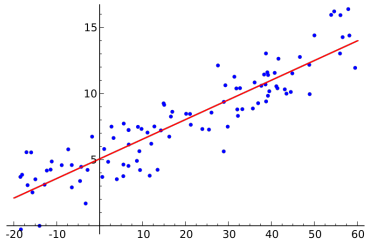
27

Correlation

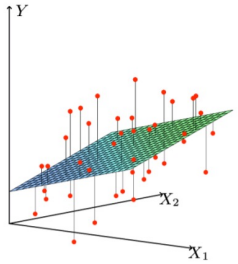
```
>>> correlations =
      df.corr()['density'].drop(['quality', 'density'])
>>> print(correlations)

fixed acidity           0.265331
volatile acidity        0.027114
citric acid             0.149503
residual sugar          0.838966
chlorides               0.257211
free sulfur dioxide     0.294210
total sulfur dioxide    0.529881
pH                     -0.093591
sulphates               0.074493
alcohol                -0.780138
Name: density, dtype: float64
```

28

Linear Regression



1 variable



2 variables

29

Linear Regression

- The predictor (independent) variable X
- The target (dependent) variable y

$$\hat{y} = \beta + \omega_1 \cdot \mathrm{X}$$

- β – the **intercept** (in ML called bias, *b*)
- ω_1 – the **slope** (in ML called weight, *w*)

- **Fit –** given pairs of data points $(\mathrm{X_i}, \mathrm{y_i})$ estimate (β, ω_1)
- **Predict –** given a new observation $\mathrm{x_{new}}$ find $\mathrm{y_{new}}$

$$\hat{y}_{new} = \beta + \omega_1 \cdot \mathrm{X}_{new}$$

30

Linear Regression

- To get the best weights, we minimize the sum of squared residuals (SSR) for all observations 1, *N* (i.e., Loss function):

$$L = \frac{1}{2}\sum_{i=1}^{N}(y_i - \beta - \omega_1 \cdot \mathrm{X}_1)^2$$

31

$$L = \frac{1}{2}\sum_{i=1}^{N}(y_i - \beta - \omega_1 \cdot \mathrm{X}_1)^2$$

$$\frac{\partial L}{\partial \omega_1} = \frac{1}{2}\sum_{i=1}^{N} 2 \cdot (y_i - \beta - \omega_1 \cdot \mathrm{X}_1) \cdot (-\mathrm{X}_1)$$

$$= -\sum_{i=1}^{N}(y_i - \beta - \omega_1 \cdot \mathrm{X}_1) \cdot \mathrm{X}_1$$

$$\frac{\partial L}{\partial \beta} = \frac{1}{2}\sum_{i=1}^{N} 2 \cdot (y_i - \beta - \omega_1 \cdot \mathrm{X}_1) \cdot (-1)$$

$$= -\sum_{i=1}^{N}(y_i - \beta - \omega_1 \cdot \mathrm{X}_1)$$

See linear_regression.ipynb for implementation of this!

32

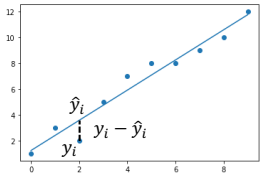
Evaluating Regression

- The quality of the regression is given by

Observed Predicted

$$R^2 = 1 - \frac{(y_i - \hat{y}_i)^2}{(y_i - y)^2}$$

Mean



https://scikit-learn.org/stable/modules/model_evaluation.html

33

Evaluating Regression

- Can we trust the R^2 value? Calculate the P-value
  - A p-value is **a measure of the probability that an observed difference could have occurred just by random chance**. The lower the p-value, the greater the statistical significance of the observed difference.
- The P-value determines the significance of the results
- If the P-Value is less than the significance level (usually 0.05) then your model fits the data well
- If the P-value is greater than the significance level then your model might not fit the data well

https://scikit-learn.org/stable/modules/model_evaluation.html

34

Linear Regression

- Linear regression is probably one of the most important and widely used regression techniques.
- Linear regression is among the simplest regression methods.
- One of its main advantages is the ease of interpreting results.

35

Linear Regression – Split data

```
>>> from sklearn.model_selection import train_test_split

>>> X = df[['alcohol']]
>>> y = df['density']

>>> X_train, X_test, y_train, y_test =
 train_test_split(X, y, test_size=0.33, random_state=42)
```

regression.ipynb

36

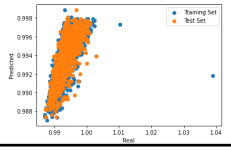
Linear Regression – Regress

```
>>> from sklearn.linear_model import LinearRegression

>>> linear_regression = LinearRegression()
>>> linear_regression.fit(X = X_train, y = y_train)
>>> print('β = ' + str(linear_regression.coef_) + ', ω1
          = ' + str(linear_regression.intercept_))
β = [-0.00190561], ω1 = 1.01408716780891
```

37

Linear Regression – Evaluate

```
>>> from sklearn.metrics import r2_score

>>> y_pred_test = linear_regression.predict(X_test)
>>> y_pred_train = linear_regression.predict(X_train)
>>> print('R2 train = ', r2_score(y_train, y_pred_train))
R2 train =  0.5926508722924497
>>> print('R2 test = ', r2_score(y_test, y_pred_test))
R2 test =  0.6447778031152053
>>> plt.scatter(y_train,y_pred_train, label='Training Set')
>>> plt.scatter(y_test,y_pred_test, label='Test Set')
>>> plt.xlabel('Real')
>>> plt.ylabel('Predicted')
>>> plt.legend()
```



38

Multiple Linear Regression

- Multiple linear regression is a generalization of what we saw before:

$$\hat{y} = \beta + \omega_1 \cdot \mathrm{X}_1 + \omega_2 \cdot \mathrm{X}_2 + \ldots + \omega_M \cdot \mathrm{X}_M$$
$$= \beta + \sum_{j=1}^{M} \omega_j \cdot \mathrm{X}_j$$

- To get the best weights, we minimize the sum of squared residuals (SSR) for all observations 1, *N* (Loss function):

$$L = \frac{1}{2} \sum_{i=1}^{N} \left(y_i - \beta - \sum_{j=1}^{M} \omega_j \cdot \mathrm{X}_{ij} \right)^2$$

# of observations — # of descriptors (variables)

39

Multiple Linear Regression – fit

```
>>> from sklearn.model_selection import train_test_split

>>> X = df.drop(['quality', 'density', 'hue'], axis=1)
>>> y = df['density']

>>> X_train, X_test, y_train, y_test =
  train_test_split(X, y, test_size=0.33, random_state=42)
>>> multiple_linear_regression = LinearRegression()
>>> multiple_linear_regression.fit(X = X_train, y = y_train)
```

40

Multiple Linear Regression – eval

```
>>> pred_train_lr = multiple_linear_regression.predict(X_train)
>>> pred_test_lr = multiple_linear_regression.predict(X_test)

>>> print('R2 training = ', r2_score(y_train, pred_train_lr))
R2 training =  0.9608334119853045
>>> print('R2 test = ', r2_score(y_test, pred_test_lr))
R2 test =  0.9724221101555639
```

41

Multiple Linear Regression – eval

- We can also use root-mean-square-error (RMSE) test:

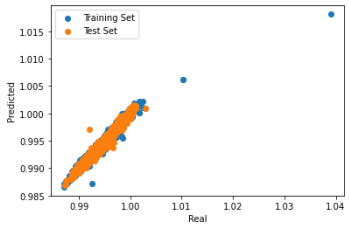
$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}$$

```
>>> from sklearn.metrics import mean_squared_error

>>> rmse_test = np.sqrt(mean_squared_error(y_test, pred_test_lr))
>>> print('RSME test= ', rmse_test)
RSME test=  0.00047624213322230194
```

42

Multiple Linear Regression – eval

```
>>> plt.scatter(y_train,pred_train_lr, label='Training Set')
>>> plt.scatter(y_test,pred_test_lr, label='Test Set')

>>> plt.xlabel('Real')
>>> plt.ylabel('Predicted')
>>> plt.legend()
```

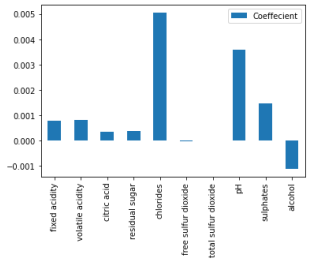


43

Multiple Linear Regression – eval

```
>>> coefficients =
  pd.DataFrame(multiple_linear_regression.coef_, X.columns.tolist())
>>> coefficients.columns = ['Coefficients']
>>> print(coefficients)
                      Coefficients
fixed acidity             0.000784
volatile acidity          0.000808
citric acid               0.000347
residual sugar            0.000380
chlorides                 0.005043
free sulfur dioxide      -0.000007
total sulfur dioxide      0.000004
pH                        0.003579
sulphates                 0.001487
alcohol                  -0.001112
>>> coefficients.plot.bar()
```

Transfer column headings of X to indeces for dataframe



44

Additional Regression Methods

- There are several additional regression methods
- Two of the most common additional regression methods are
  - LASSO
  - Ridge
  - These are both so-called "shrinkage methods" as they shrink coefficients towards zero, to generate more sparse models
  - These models are easier to interpret

45

LASSO

- Least Absolute Shrinkage and Selection Operator (LASSO) is a linear regression method that produces variable selection and regularization to improve the prediction accuracy and generate a smaller model. This method uses a cost function with a constant λ that defines the degree of penalization.
- Minimize the RSS (Loss function):

$$L = \sum_{i=1}^{N}\left(y_i - \beta - \sum_{j=1}^{M}\omega_j \cdot \mathrm{X}_{ij}\right)^2 + \lambda \cdot \sum_{j=1}^{M}|\omega_j|$$

- Some of the ω_j s are shrunk to exactly zero, resulting in a regression model that's easier to interpret.
- If there is a group of highly correlated variables, then the LASSO method tends to randomly select one variable from a group and ignore the others.

46

LASSO

- When λ=0, no parameters are eliminated. The estimate is equal to the one found with linear regression.
- As λ increases, more and more coefficients are set to zero and eliminated (theoretically, when $\lambda = \infty$, all coefficients are eliminated).
- As λ increases, bias increases.
- As λ decreases, variance increases.

$$L = \sum_{i=1}^{N}\left(y_i - \beta - \sum_{j=1}^{M}\omega_j \cdot \mathrm{X}_{ij}\right)^2 + \lambda \cdot \sum_{j=1}^{M}|\omega_j|$$

47

Ridge

- Ridge is a linear regression method that produces variable selection and regularization to improve the prediction accuracy and generate a smaller model. This method uses a cost function with a constant λ that defines the degree of penalization.
- Minimize the RSS (Loss function):

$$L = \sum_{i=1}^{N}\left(y_i - \beta - \sum_{j=1}^{M}\omega_j \cdot \mathrm{X}_{ij}\right)^2 + \lambda \cdot \sum_{j=1}^{M}{\omega_j}^2$$

- Some of the ω_j s are shrunk close to zero, resulting in a regression model that's easier to interpret.

48

Ridge

- When λ=0, no parameters are eliminated. The estimate is equal to the one found with linear regression.
- As λ increases, more and more coefficients are set to zero and eliminated (theoretically, when $\lambda = \infty$, all coefficients are eliminated).
- As λ increases, bias increases.
- As λ decreases, variance increases.

$$L = \sum_{i=1}^{N}\left(y_i - \beta - \sum_{j=1}^{M}\omega_j \cdot \mathrm{X}_{ij}\right)^2 + \lambda \cdot \sum_{j=1}^{M}{\omega_j}^2$$

49

Comparison LASSO vs. Ridge

- Typical Use Cases
  - Ridge: Prevent overfitting. Since it includes all the features, it is not very useful in case of exorbitantly high # of features (e.g., say in millions).
  - LASSO: Prevent overfitting. Since it provides sparse solutions, it is generally the model of choice (or some variant of this concept) for modelling cases where the # of features are in millions or more. Easier to interpret.
- Presence of Highly Correlated Features
  - Ridge: It generally works well even in presence of highly correlated features as it will include all of them in the model, but the coefficients will be distributed among them depending on the correlation.
  - LASSO: It arbitrarily selects any one feature among the highly correlated ones and reduced the coefficients of the rest to zero. Also, the chosen variable changes randomly with change in model parameters.
- Along with Ridge and Lasso, Elastic Net is another useful techniques which combines features of both (L1 and L2 regularization). It can be used to balance out the pros and cons of ridge and lasso regression.

50

LASSO Example

```
>>> from sklearn.linear_model import Lasso

>>> lasso_regression = Lasso(alpha=0.0001) # Try with 0.0 and 10000.0
>>> lasso_regression.fit(X = X_train, y = y_train)

>>> from sklearn.metrics import r2_score, mean_squared_error

>>> y_pred = lasso_regression.predict(X_test)
>>> r2 = r2_score(y_test, y_pred)
>>> print('R2 test = ', r2)
R2 test =  0.9425029466919721
```

51

LASSO Example

```
>>> coeffecients =
    pd.DataFrame(lasso_regression.coef_, X.columns.tolist())
>>> coeffecients.columns = ['Coeffecient']
>>> print(coeffecients)
```

Transfer column headings of X to indeces for dataframe

	Coeffecient
fixed acidity	0.000383
volatile acidity	0.000000
citric acid	0.000000
residual sugar	0.000362
chlorides	0.000000
free sulfur dioxide	-0.000011
total sulfur dioxide	0.000009
pH	0.000000
sulphates	0.000000
alcohol	-0.001029

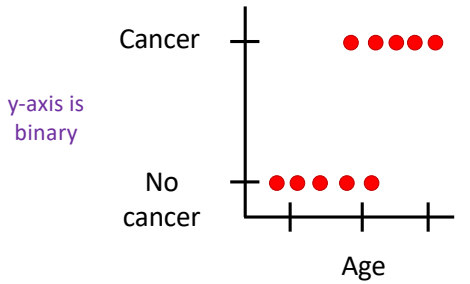
52

Logistic Regression

- Logistic regression is one of the simplest and commonly used methods for solving the binary classification problem.
- Logistic regression is easy to implement and can be used as the baseline for any binary classification problem.
- Logistic Regression can be used for various classification problems such as:
  - Spam detection.
  - Diabetes / Cancer prediction.
  - Churn detection – if a given customer will purchase a particular product or will they churn another competitor.
  - Click prediction - whether the user will click on a given advertisement link or not.

53

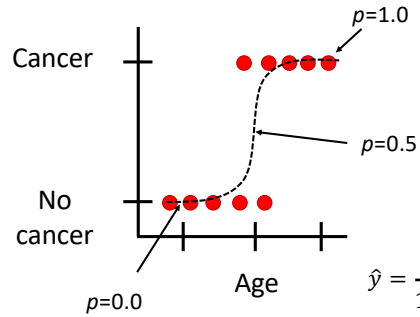
Logistic Regression



54

Logistic Regression

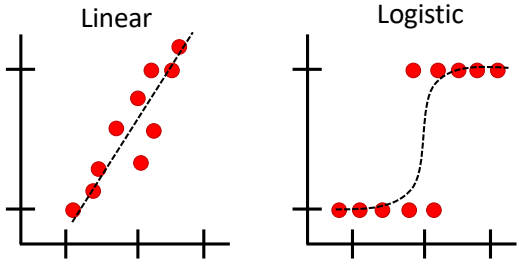
- Logistic regression fits a sigmoid function (S-shaped logistic function)



$$\hat{y} = \frac{1}{1 + e^{-(\beta+\omega_1 \cdot \mathrm{X})}}$$

55

Logistic Regression vs Linear Regression



$$\hat{y} = \beta + \omega_1 \cdot \mathrm{X}$$

$$\hat{y} = \frac{1}{1 + e^{-(\beta+\omega_1 \cdot \mathrm{X})}}$$

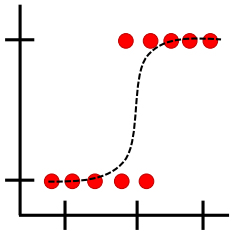
56

Logistic Regression vs Linear Regression

- Linear regression outputs a continuous value, while logistic regression outputs a constant value (0 or 1).
  - An example of the continuous output is house price and stock price. Examples of the discrete output is predicting whether a patient has cancer or not, predicting whether the customer will churn.
- Both can be used with multiple variables (X)
- Linear regression is estimated using Ordinary Least Squares (OLS) while logistic regression is estimated using Maximum Likelihood Estimation (MLE) approach.

57

Logistic Regression and maximum likelihood

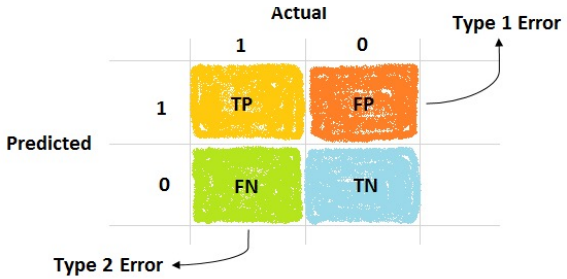


Calculate the probability of finding a point along the S-curve and find the curve that matches the probabilities of all the points best.

58

Evaluating Classification

- The quality of a classification may be estimated by the confusion matrix



https://scikit-learn.org/stable/modules/model_evaluation.html

59

Evaluating Classification

- Precision

$$P = \frac{TP}{TP + FP}$$

True positive

False positive (i.e., should have been negative)

  - How many of the correctly predicted cases actually turned out to be positive?
- Recall

$$R = \frac{TP}{TP + FN}$$

False negative (i.e., should have been positive)

  - How many of the actual positive cases we were able to predict correctly with our model?

https://scikit-learn.org/stable/modules/model_evaluation.html

60

Evaluating Classification

- The quality of a classification is given by

$$F_1 = \frac{2}{\frac{1}{R}+\frac{1}{P}} = \frac{TP}{TP+\frac{1}{2}(FP+FN)}$$

- The F1-score captures both the trends in a single value
- F1-score is a harmonic mean of Precision and Recall, and so it gives a combined idea about these two metrics. It is maximum when Precision is equal to Recall.
- F1 Score becomes 1 only when precision and recall are both 1.
- F1 score becomes high only when both precision and recall are high.

https://scikit-learn.org/stable/modules/model_evaluation.html

61

Multiple Logistic Regression Example

- We will use all the wine features with logistic regression to classify white and red wine.

```
>>> from sklearn.linear_model import LogisticRegression
>>> from sklearn.metrics import classification_report,
                                accuracy_score, confusion_matrix

>>> lr = LogisticRegression(random_state=0)
>>> lr.fit(X_train, y_train)
>>> # Now we'll test the data
>>> y_pred = lr.predict(X_test)
>>> print("Accuracy score: " + str(accuracy_score(y_test, y_pred)))
>>> print("\nConfusion matrix: \n" + str(confusion_matrix(y_test, y_pred)))
>>> print("\nClassification report: \n" + str(classification_report(y_test,
            y_pred)))
```

logistic_regression.ipynb

62

```
Accuracy score: 0.9888111888111888

Confusion matrix:
[[ 544   13]
 [  11 1577]]

Classification report:
              precision    recall  f1-score   support

           0       0.98      0.98      0.98       557
           1       0.99      0.99      0.99      1588

    accuracy                           0.99      2145
   macro avg       0.99      0.98      0.99      2145
weighted avg       0.99      0.99      0.99      2145
```

logistic_regression.ipynb

63

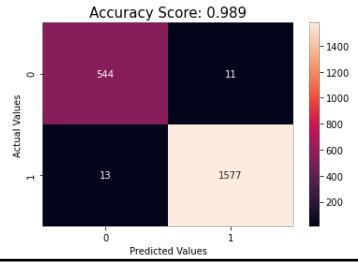
```
>>> train_accuracy = lr.score(X_train, y_train)
>>> test_accuracy = lr.score(X_test, y_test)
>>> print('One-vs-rest', '-'*35,
      'Accuracy in Train Group   : {:.3f}'.format(train_accuracy),
      'Accuracy in Test  Group   : {:.3f}'.format(test_accuracy),
       sep='\n')
```

```
One-vs-rest
-----------------------------------
Accuracy in Train Group   : 0.995
Accuracy in Test  Group   : 0.989
```

logistic_regression.ipynb

64

```
>>> from sklearn.metrics import confusion_matrix as cm
>>> predictions = lr.predict(X_test)
>>> score = round(accuracy_score(y_test, predictions), 3)
>>> cm1 = cm(predictions, y_test)
>>> sns.heatmap(cm1, annot=True, fmt=".0f")
>>> plt.xlabel('Predicted Values')
>>> plt.ylabel('Actual Values')
>>> plt.title('Accuracy Score: {0}'.format(score), size = 15)
>>> plt.show()
```



logistic_regression.ipynb

65

```
>>> from sklearn.metrics import precision_score
>>> print("precision score         : ",  precision_score(y_test,
          predictions, average='micro'))
precision score         :  0.9888111888111888
>>> from sklearn.metrics import recall_score
>>> print("recall score            : ",  recall_score(y_test, predictions,
          average='micro'))
recall score            :  0.9888111888111888
>>> from sklearn.metrics import f1_score
>>> precision_s = precision_score(y_test, predictions,average='micro')
>>> recall_s    = recall_score(y_test, predictions, average='micro')
>>> print("F1_score     : ",  2/((1/precision_s) + (1/precision_s)))
F1_score     :  0.9888111888111889
>>> print("F1_score     : ", f1_score(y_test, predictions,average='micro'))
F1_score     :  0.9888111888111888
```

logistic_regression.ipynb

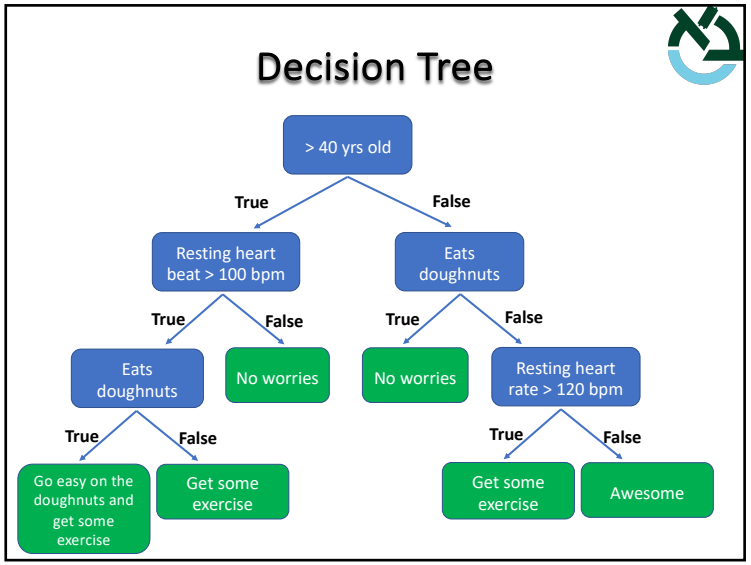
66

67

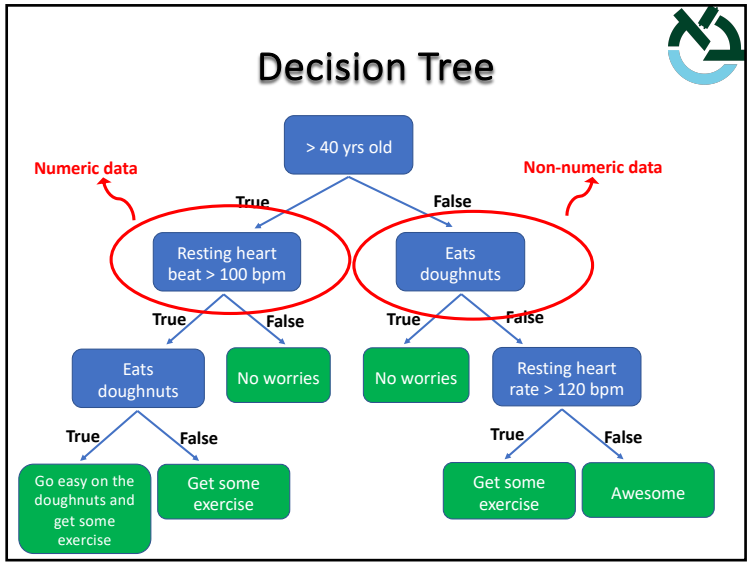
Decision Tree

- Decision trees are a non-parametric supervised learning method used for classification and regression.
- The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.
- Decision Tree example for the Iris dataset https://scikit-learn.org/stable/auto_examples/tree/plot_iris_dtc.html

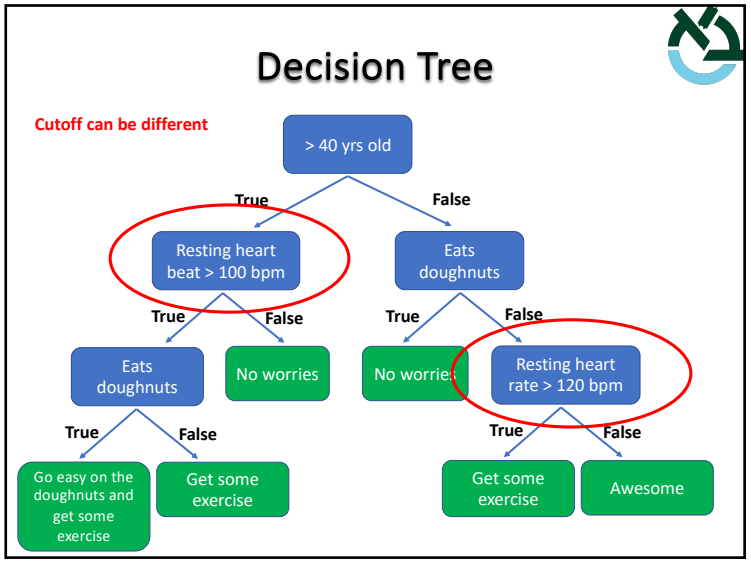
68


Decision Tree
> 40 yrs old
True
False
Resting heart beat > 100 bpm
Eats doughnuts
True
False
True
False
Eats doughnuts
No worries
No worries
Resting heart rate > 120 bpm
True
False
True
False
Go easy on the doughnuts and get some exercise
Get some exercise
Get some exercise
Awesome

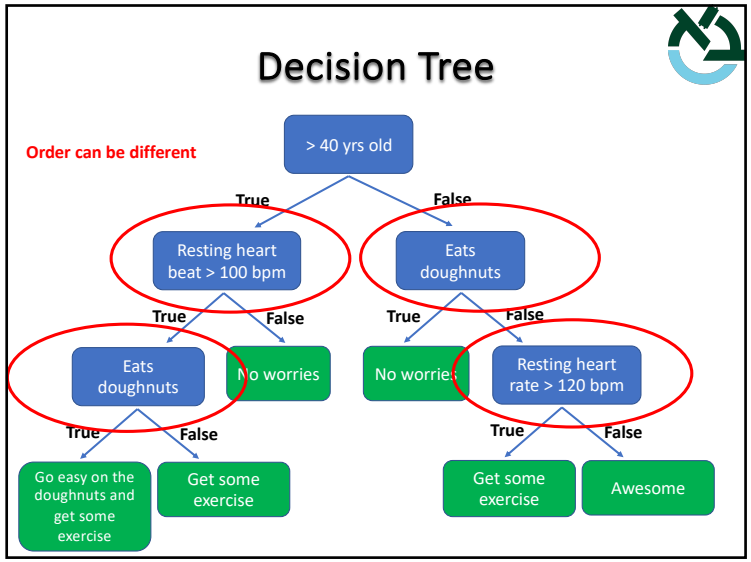
69


Decision Tree
Numeric data
Non-numeric data
> 40 yrs old
True
False
Resting heart beat > 100 bpm
Eats doughnuts
True
False
True
False
Eats doughnuts
No worries
No worries
Resting heart rate > 120 bpm
True
False
True
False
Go easy on the doughnuts and get some exercise
Get some exercise
Get some exercise
Awesome

70

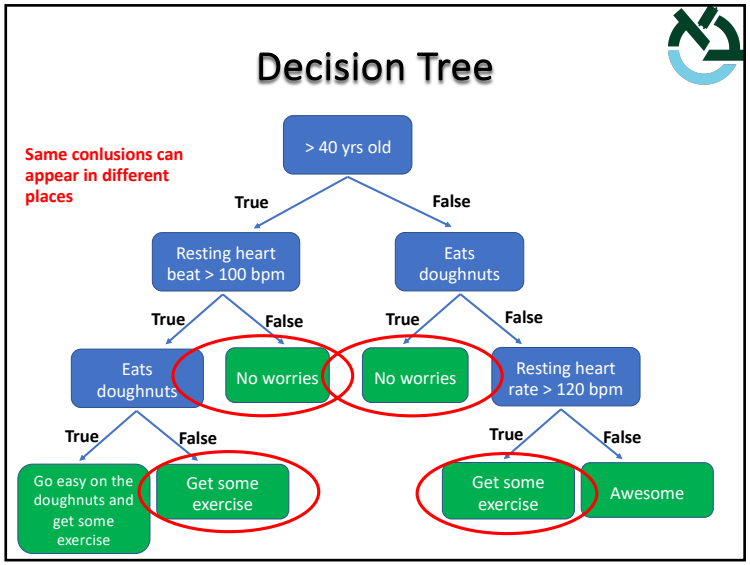

Decision Tree
Cutoff can be different
> 40 yrs old
True
False
Resting heart beat > 100 bpm
Eats doughnuts
True
False
True
False
Eats doughnuts
No worries
No worries
Resting heart rate > 120 bpm
True
False
True
False
Go easy on the doughnuts and get some exercise
Get some exercise
Get some exercise
Awesome

71


Decision Tree
Order can be different
> 40 yrs old
True
False
Resting heart beat > 100 bpm
Eats doughnuts
True
False
True
False
Eats doughnuts
No worries
No worries
Resting heart rate > 120 bpm
True
False
True
False
Go easy on the doughnuts and get some exercise
Get some exercise
Get some exercise
Awesome

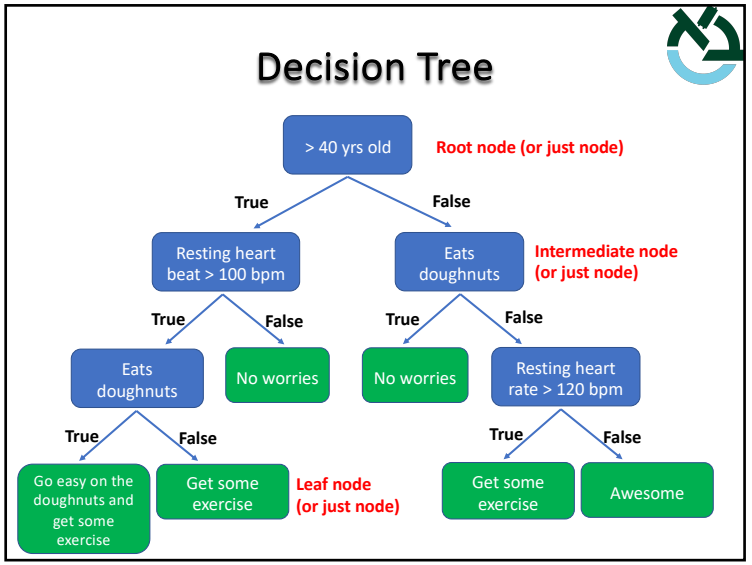
72

Decision Tree



73

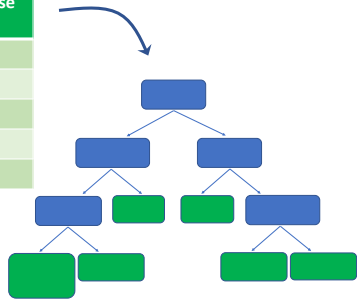
Decision Tree



74

Raw data → Decision Tree

Chest pain	Good blood circulation	Blocked arteries	Heart disease
No	No	No	No
Yes	Yes	Yes	Yes
Yes	Yes	No	No
Yes	No	???	Yes
...	...	...	...



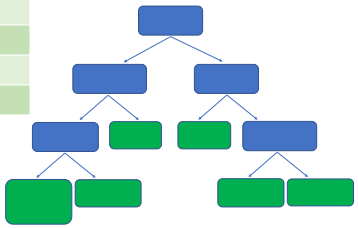
We want to use medical features to predict whether or not a patient has heart disease

75

Decision Tree

Chest pain	Good blood circulation	Blocked arteries	Heart disease
No	No	No	No
Yes	Yes	Yes	Yes
Yes	Yes	No	No
Yes	No	???	Yes
...	...	...	...

Should Chest pain, Good blood circulation or Blocked arteries be at the top of the tree?

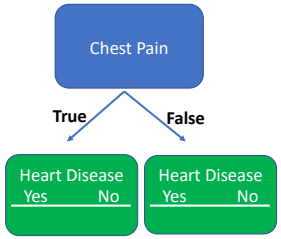


76

Decision Tree

Chest pain	Good blood circulation	Blocked arteries	Heart disease
No	No	No	No
Yes	Yes	Yes	Yes
Yes	Yes	No	No
Yes	No	???	Yes
...	...	...	...

How well does chest pain alone predict heart disease?



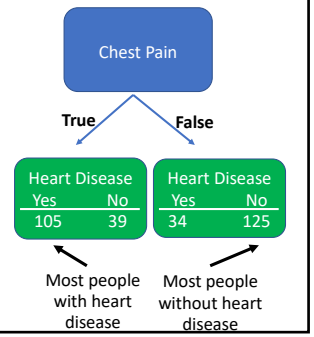
77

Decision Tree

Chest pain	Good blood circulation	Blocked arteries	Heart disease
No	No	No	No
Yes	Yes	Yes	Yes
Yes	Yes	No	No
Yes	No	???	Yes
...	...	...	...

303 patients

How well does chest pain alone predict heart disease?



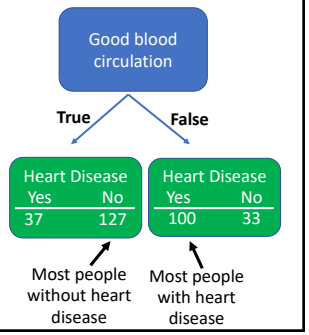
78

Decision Tree

Chest pain	Good blood circulation	Blocked arteries	Heart disease
No	No	No	No
Yes	Yes	Yes	Yes
Yes	Yes	No	No
Yes	No	???	Yes
...	...	...	...

303 patients

How well does good blood circulation alone predict heart disease?



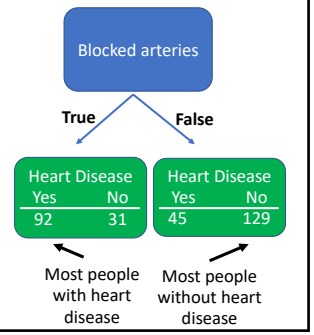
79

Decision Tree

Chest pain	Good blood circulation	Blocked arteries	Heart disease
No	No	No	No
Yes	Yes	Yes	Yes
Yes	Yes	No	No
Yes	No	???	Yes
...	...	...	...

303 patients (skip the ones missing data or use some method to fill in)

How well does chest pain alone predict heart disease?

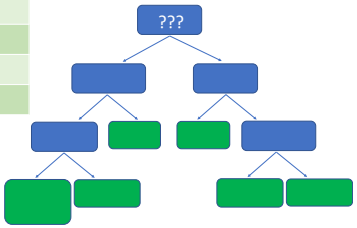


80

Decision Tree

Chest pain	Good blood circulation	Blocked arteries	Heart disease
No	No	No	No
Yes	Yes	Yes	Yes
Yes	Yes	No	No
Yes	No	???	Yes
...	...	...	...

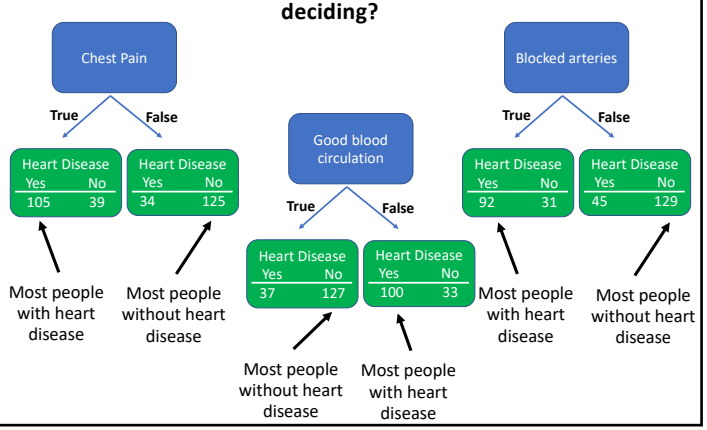
Which feature should be at the root node?



81

Decision Tree

Which feature is best at deciding?

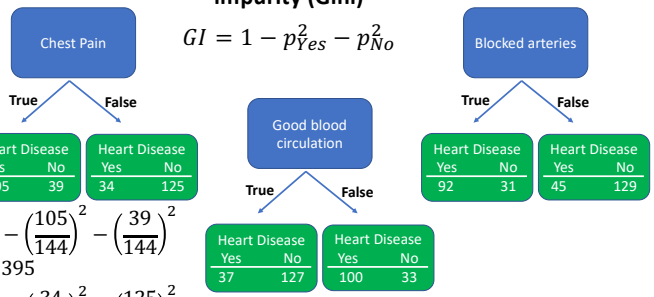


82

Decision Tree

Quantify by calculating impurity (Gini)

$$GI = 1 - p_{Yes}^2 - p_{No}^2$$



$$GI = 1 - \left(\frac{105}{144}\right)^2 - \left(\frac{39}{144}\right)^2 = 0.395$$

$$GI = 1 - \left(\frac{34}{159}\right)^2 - \left(\frac{125}{159}\right)^2 = 0.336$$

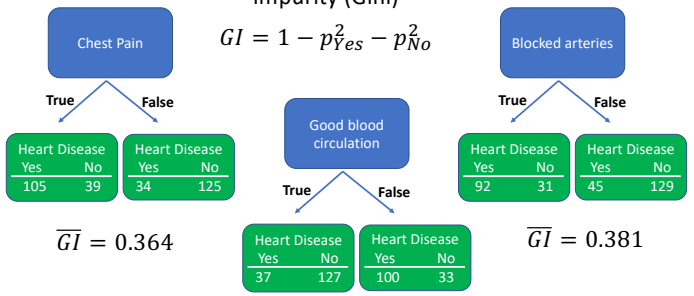
$$\overline{GI} = \left(\frac{144}{144+159}\right)0.395 - \left(\frac{159}{144+159}\right)0.336 = 0.364$$

83

Decision Tree

Quantify by calculating impurity (Gini)

$$GI = 1 - p_{Yes}^2 - p_{No}^2$$



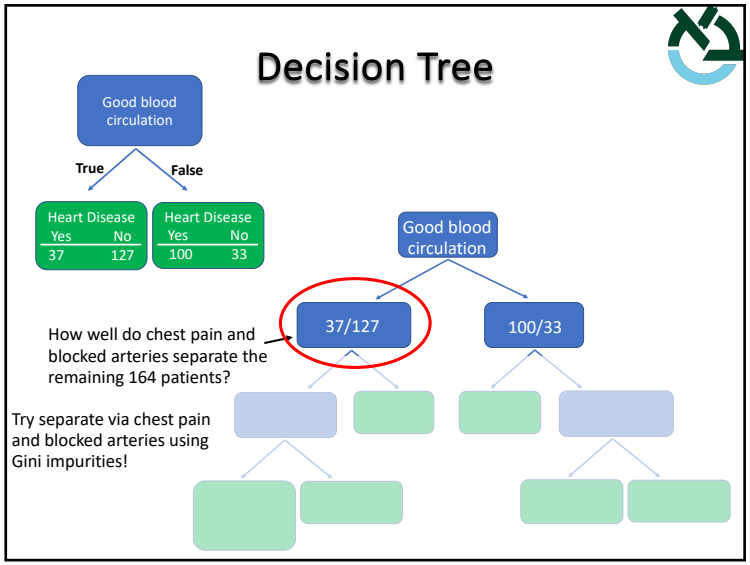
$\overline{GI} = 0.364$

$\overline{GI} = 0.381$

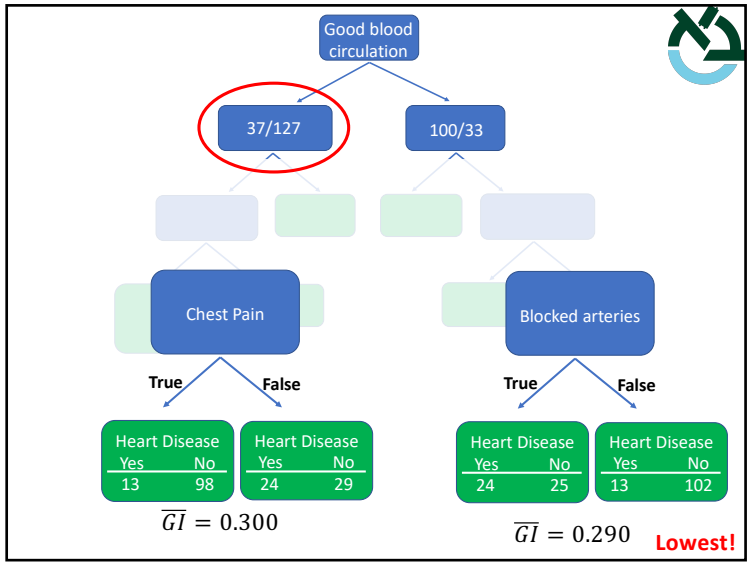
$\overline{GI} = 0.360$

Lowest impurity, i.e., separates heart disease (yes/no) the best → Leaf node

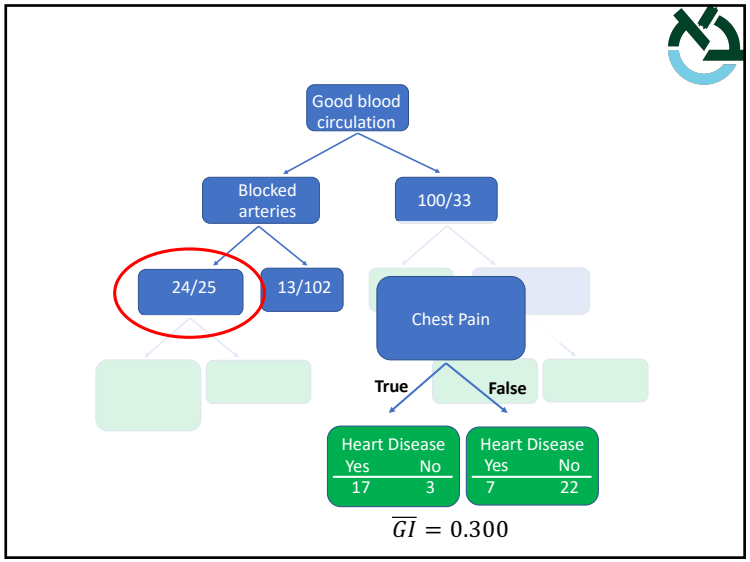
84


Decision Tree
Good blood circulation
True
False
Heart Disease
Yes No
37 127
Heart Disease
Yes No
100 33
Good blood circulation
37/127
100/33
How well do chest pain and blocked arteries separate the remaining 164 patients?
Try separate via chest pain and blocked arteries using Gini impurities!

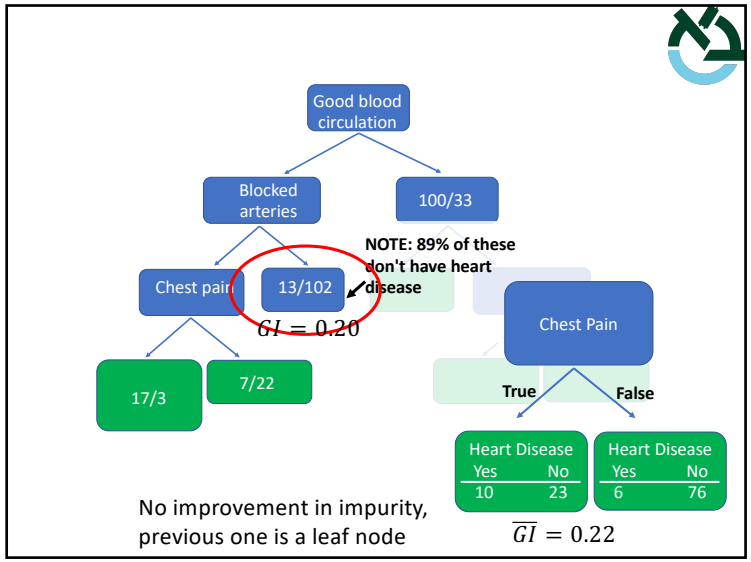
85


Good blood circulation
37/127
100/33
Chest Pain
Blocked arteries
True
False
Heart Disease
Yes No
13 98
Heart Disease
Yes No
24 29
Heart Disease
Yes No
24 25
Heart Disease
Yes No
13 102
$\overline{GI} = 0.300$
$\overline{GI} = 0.290$
Lowest!

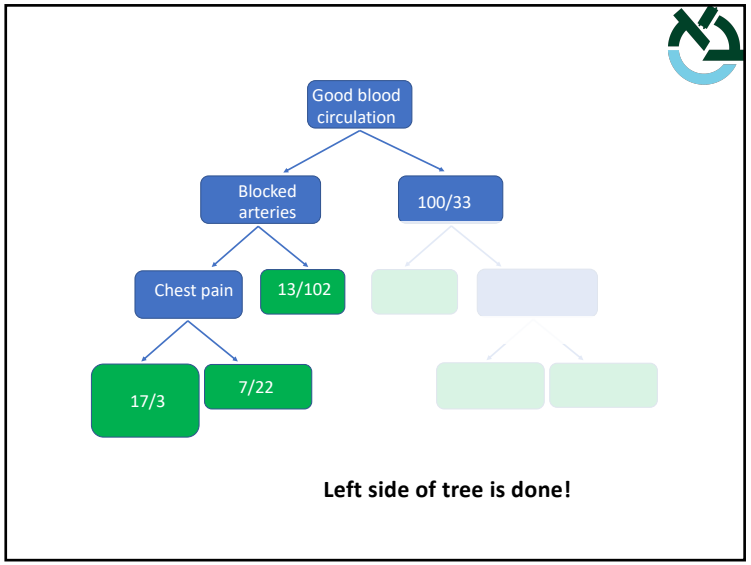
86


Good blood circulation
Blocked arteries
100/33
24/25
13/102
Chest Pain
True
False
Heart Disease
Yes No
17 3
Heart Disease
Yes No
7 22
$\overline{GI} = 0.300$

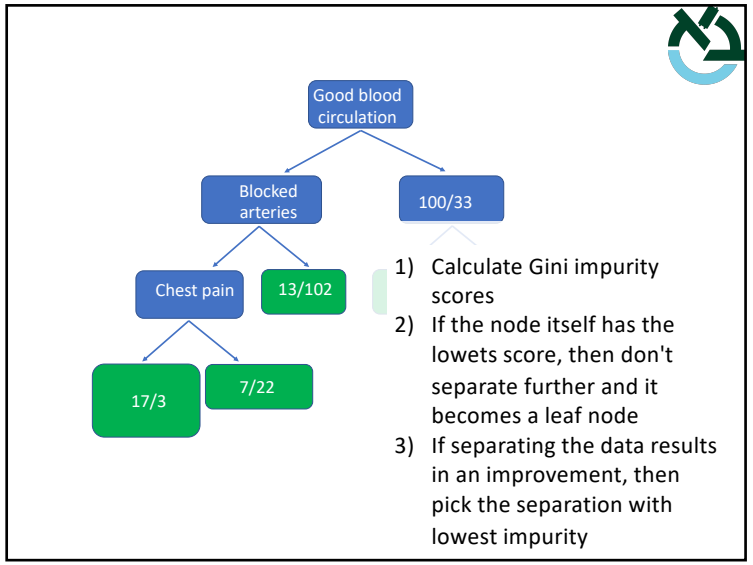
87


Good blood circulation
Blocked arteries
100/33
NOTE: 89% of these don't have heart disease
Chest pain
13/102
$GI = 0.20$
17/3
7/22
Chest Pain
True
False
Heart Disease
Yes No
10 23
Heart Disease
Yes No
6 76
No improvement in impurity, previous one is a leaf node
$\overline{GI} = 0.22$

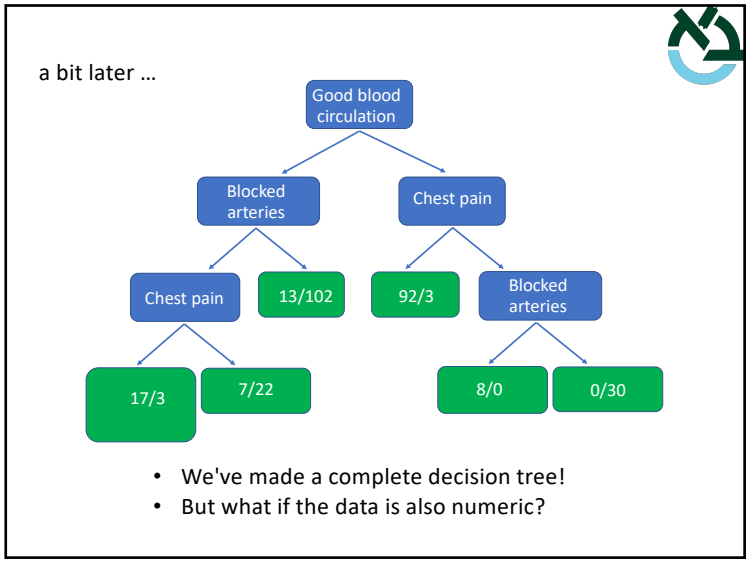
88


Good blood circulation
Blocked arteries
100/33
Chest pain
13/102
17/3
7/22
Left side of tree is done!

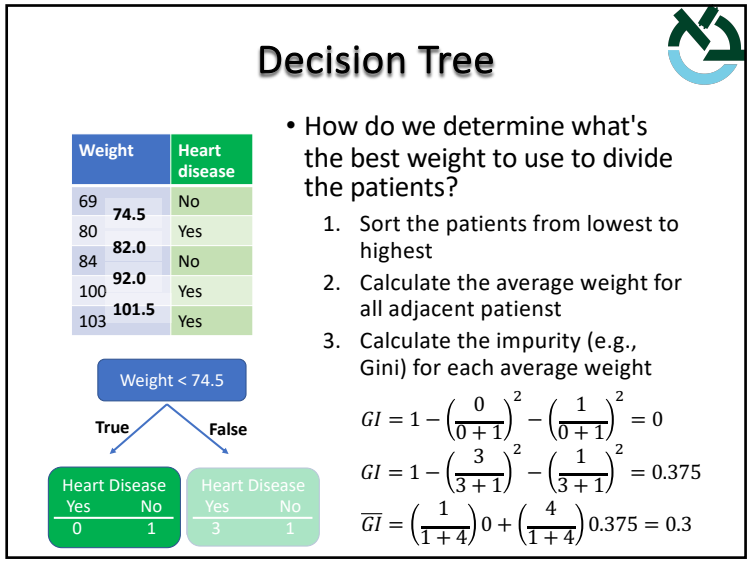
89


Good blood circulation
Blocked arteries
100/33
Chest pain
13/102
17/3
7/22
1) Calculate Gini impurity scores
2) If the node itself has the lowets score, then don't separate further and it becomes a leaf node
3) If separating the data results in an improvement, then pick the separation with lowest impurity

90


a bit later …
Good blood circulation
Blocked arteries
Chest pain
Chest pain
13/102
92/3
Blocked arteries
17/3
7/22
8/0
0/30
• We've made a complete decision tree!
• But what if the data is also numeric?

91


Decision Tree
Weight
Heart disease
69
74.5
80
No
Yes
82.0
84
No
92.0
100
Yes
101.5
103
Yes
Weight < 74.5
True
False
Heart Disease
Yes
No
0
1
Heart Disease
Yes
No
3
1
• How do we determine what's the best weight to use to divide the patients?
1. Sort the patients from lowest to highest
2. Calculate the average weight for all adjacent patienst
3. Calculate the impurity (e.g., Gini) for each average weight
$GI = 1 - \left(\frac{0}{0+1}\right)^2 - \left(\frac{1}{0+1}\right)^2 = 0$
$GI = 1 - \left(\frac{3}{3+1}\right)^2 - \left(\frac{1}{3+1}\right)^2 = 0.375$
$\overline{GI} = \left(\frac{1}{1+4}\right)0 + \left(\frac{4}{1+4}\right)0.375 = 0.3$

92

Decision Tree

Weight	Heart disease
69	No
74.5	
80	Yes
82.0	
84	No
92.0	
100	Yes
101.5	
103	Yes

Weight < 82.0

True False

Heart Disease	
Yes	No
1	1

Heart Disease	
Yes	No
2	1

$$GI = 1 - \left(\frac{1}{1+1}\right)^2 - \left(\frac{1}{1+1}\right)^2 = 0.5$$

$$GI = 1 - \left(\frac{2}{2+1}\right)^2 - \left(\frac{1}{2+1}\right)^2 = 0.44$$

$$\overline{GI} = \left(\frac{2}{2+3}\right)0.5 + \left(\frac{3}{2+3}\right)0.44 = 0.47$$

93

Decision Tree

Weight	Heart disease
69	No
74.5	
80	Yes
82.0	
84	No
92.0	
100	Yes
101.5	
103	Yes

Weight < 92.0

True False

Heart Disease	
Yes	No
1	2

Heart Disease	
Yes	No
2	0

$$GI = 1 - \left(\frac{1}{1+2}\right)^2 - \left(\frac{2}{1+2}\right)^2 = 0.44$$

$$GI = 1 - \left(\frac{2}{2+0}\right)^2 - \left(\frac{0}{2+0}\right)^2 = 0$$

$$\overline{GI} = \left(\frac{3}{3+2}\right)0.44 + \left(\frac{2}{3+2}\right)0 = 0.267$$

94

Decision Tree

Weight	Heart disease
69	No
74.5	
80	Yes
82.0	
84	No
92.0	
100	Yes
101.5	
103	Yes

Weight < 101.5

True False

Heart Disease	
Yes	No
2	2

Heart Disease	
Yes	No
1	0

$$GI = 1 - \left(\frac{2}{2+2}\right)^2 - \left(\frac{2}{2+2}\right)^2 = 0.44$$

$$GI = 1 - \left(\frac{1}{1+0}\right)^2 - \left(\frac{0}{1+0}\right)^2 = 0$$

$$\overline{GI} = \left(\frac{4}{4+1}\right)0.5 + \left(\frac{1}{4+1}\right)0 = 0.4$$

95

Decision Tree

Weight	Heart disease	
69	No	$\overline{GI} = 0.3$
74.5		
80	Yes	$\overline{GI} = 0.47$
82.0		
84	No	$\overline{GI} = 0.27$
92.0		
100	Yes	$\overline{GI} = 0.4$
101.5		
103	Yes	

- The lowest impurity (0.27) occurs when we separate using weight < 92.0
- This is the cutoff we'll use when we compare weight to chest pain or blocked arteries

96

Decision Tree Pros

- Simple to understand and to interpret. Trees can be visualized.
- Requires little data preparation.
- The cost of using the tree is logarithmic in the number of data points used to train the tree.
- Able to handle both numerical and categorical data.
- Able to handle multi-output problems.
- Performs ok even if its assumptions are somewhat violated by the true model from which the data were generated.

97

Decision Tree Cons

- Trees have one aspect that prevents them from being the ideal tool for predictive learning, namely inaccuracy
- Trees work great with the data used to create them, but they are not flexible when it comes to classifying new samples

98

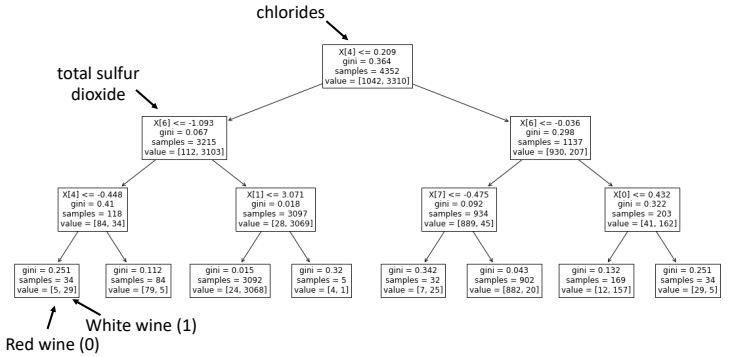
Decision Tree in action

```
>>> from sklearn import tree
>>> from sklearn.tree import DecisionTreeClassifier

>>> dt = DecisionTreeClassifier(criterion = 'gini', max_depth = 3,
                                random_state = 1)
>>> dt.fit(X_train, y_train)
>>> y_pred = dt.predict(X_test)
>>> print("Accuracy score: " + str(accuracy_score(y_test, y_pred)))
>>> print("\nConfusion matrix: \n" + str(confusion_matrix(y_test,
          y_pred)))
>>> print("\nClassification report: \n" +
          str(classification_report(y_test, y_pred)))
>>> fig = plt.subplots(figsize=(20, 8))
>>> tree.plot_tree(dt, fontsize=12)
```

trees.ipynb

99

Decision Tree in action

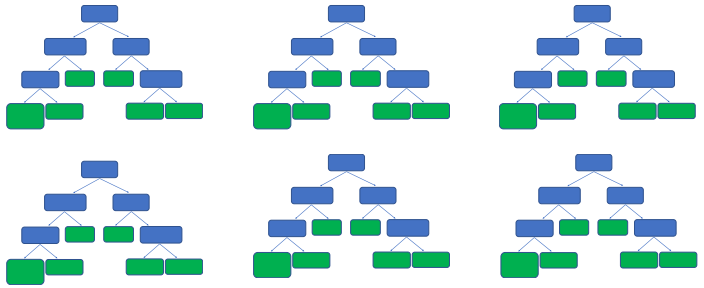


trees.ipynb

100

Decision Tree → Random Forest

- Random forests combine the simplicity of decision trees with flexibility resulting in vast improvement in accuracy



101

Random Forest

- A Random Forest (RF) combines many decision trees.
- In a classification problem, each individual decision tree in the Random Forest decides ("votes") which class to classify an input as, and the forest chooses the classification with the most "votes".
- RF fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.
- Let's make a RF …

102

Random Forest

- Create a "bootstrapped" dataset

Original dataset

Chest pain	Good blood circulation	Blocked arteries	Weight	Heart disease
No	No	No	60	No
Yes	Yes	Yes	80	Yes
Yes	Yes	No	100	No
Yes	No	Yes	75	Yes

103

Original dataset

Chest pain	Good blood circulation	Blocked arteries	Weight	Heart disease
No	No	No	60	No
Yes	Yes	Yes	80	Yes
Yes	Yes	No	100	No
Yes	No	Yes	75	Yes

Bootstrapped dataset

Chest pain	Good blood circulation	Blocked arteries	Weight	Heart disease

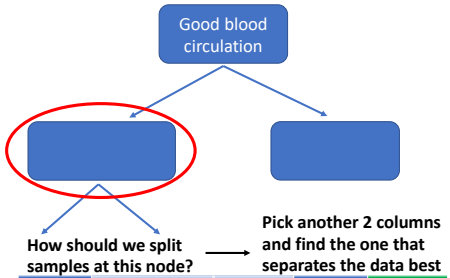
104

- Create tree from bootstrapped dataset
- Use a random subset of the variables (columns) at each step

Chest pain	Good blood circulation	Blocked arteries	Weight	Heart disease
Yes	Yes	Yes	80	Yes
No	No	No	60	No
Yes	No	Yes	75	Yes
Yes	No	Yes	75	Yes

- We select Good blood circulation and Blocked arteries as candidates for root node
- Let's assume that Good blood circulation separated data best

105



How should we split samples at this node?

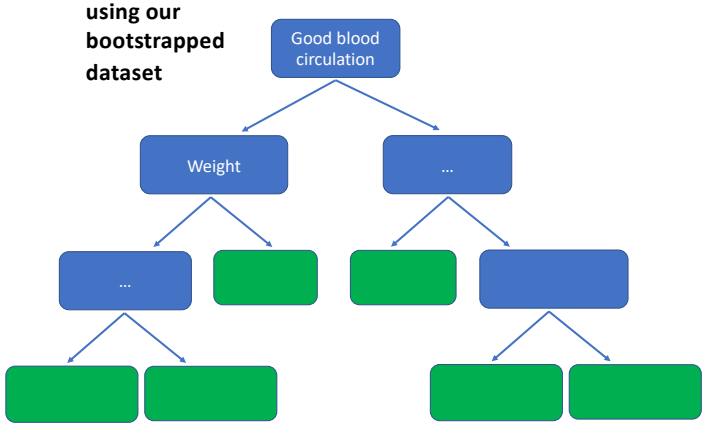
Pick another 2 columns and find the one that separates the data best

Chest pain	Good blood circulation	Blocked arteries	Weight	Heart disease
Yes	Yes	Yes	80	Yes
No	No	No	60	No
Yes	No	Yes	75	Yes
Yes	No	Yes	75	Yes

And so on…

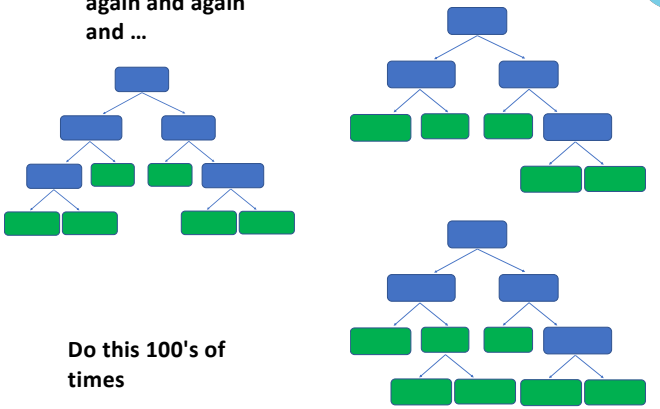
106

Here's the tree we created using our bootstrapped dataset



107

Now do this again and again and …



Do this 100's of times

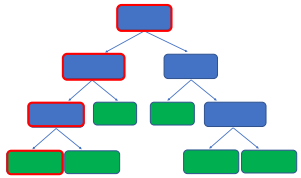
108

RF – how do we use the forest?

- We get a new patient. Does he/she have a heart disease?

Chest pain	Good blood circulation	Blocked arteries	Weight	Heart disease
Yes	Yes	Yes	78	???

- Run the data down the first tree:



1st tree says yes, heart disease

Keep track of results:

Heart Disease	
Yes	No
1	0

Keep on doing this for all trees …

109

RF – how do we use the forest?

- The final tally is

Heart Disease	
Yes	No
95	5

- Person has heart disease ☹
- This process of bootstrapping and counting is called bagging
- About 1/3 of the original data samples (rows) does not get used (by random choice). Called out-of-bag dataset

110

RF – how do we use the forest?

- How can we test whether our forest is any good?
- Run the out-of-bag dataset through the forest and see if it predicts correctly (assume 100 trees)
- Sample 1:

Classification of out-of-bag-sample	
Yes	No
7	93

Mostly right (patient does not have heart disease)

- Sample 2:

Classification of out-of-bag-sample	
Yes	No
95	5

Mostly right (patient does have heart disease)

And so on …

111

RF – how do we use the forest?

- The final measure of the accuracy of our forest is the **proportion of out-of-bag samples that were *incorrectly* classified is the "out-of-bag error"**
- Note: Remember we said we randomly choose 2 variables (columns) and find the one that separates the data best. What if we chose 3 variable? Or 4 (if there are more)?
  - Check all possible number of variables

112

RF – the complete process

1. Build a random forest
2. Estimate the accuracy of the random forest

Change the number of variables used at each step

Do this many times and choose the most accurate one

113

RF – missing data

- How do we handle missing data?
- RF considers 2 types of missing data:
- Missing data in the original data set used to create the RF

Chest pain	Good blood circulation	Blocked arteries	Weight	Heart disease
No	No	No	60	No
Yes	Yes	Yes	80	Yes
Yes	Yes	No	100	No
Yes	No	???	???	No

- Missing data in the new sample that you want to categorize

Chest pain	Good blood circulation	Blocked arteries	Weight	Heart disease
Yes	No	No	???	???

114

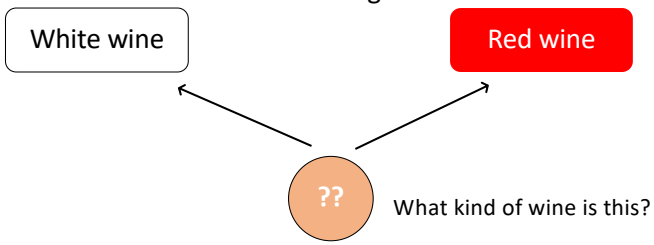
RF – missing data

- Missing data in the original data set used to create the RF
  - Make an initial guess and gradually improve guess
    - Most common value (non-numeric data)
    - Median value (numeric data)
  - Determine which samples are similar to one with missing data+guesses by running this sample through tree (similarity is defined as ending up at the same root)
    - Build proximity matrix which counts similarity between data samples
  - Improve guess of missing data sample by using most similar data sample and rerun through tree (until convergence)
- Missing data in the new sample that you want to categorize
  - Same idea …

115

kNN Classification

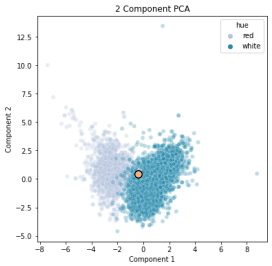
- **k-NN classification:** The objective is to classify an unknown object by finding the most common class nearest to its features.
- kNN is a simple way to classify data
- Assume we have the following datasets:



116

kNN Classification

- Step 1: Start with a dataset of known categories.
- Step 2: Cluster the data (e.g., PCA) (this is the training data for kNN)
- Step 3: Add a new sample without a label
- Step 4: Classify the new sample by looking at the nearest annotated point (i.e., NN)
  - If k=1, look at 1 nearest neighbor
  - If k=11 use 11 nearest neighbors and determine label by counting votes

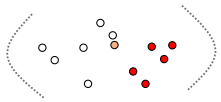


117

kNN Classification

- Example (k=11):
  - 6 nearest neighbors are white
  - 5 nearest neighbors are red

More data points here …



More data points here …

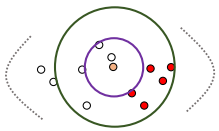
- In this case, the new wine samples has more white neighbors than red, so we classify it as white
- But what's the exact algorithm?

118

kNN Classification

1. Calculate the distance from 'x' to all points in your data.
2. Sort the points in your data by increasing distance from x.
3. Predict the majority label of the "k" closest points.
   - If k=2, the algorithm will look at the 2 nearest neighbors to this new data point (inner circle) where we have two white wine points. 'x' is classified as white wine.
   - If k=9, the algorithm will look at the 9 nearest neighbors to this new data point (outer circle). 'x' is classified as red wine.

More data points here …



More data points here …

119

kNN Classification

- **How do you calculate the distance?**
- Minkowski Distance:

$$d_{a\to b} = \left(\sum_{i=1}^{n} |a_i - b_i|^p\right)^{1/p}$$

- Manhattan distance (p=1)

$$d_{a\to b} = \left(\sum_{i=1}^{n} |a_i - b_i|\right)$$

- Euclidean distance (p=2)

$$d_{a\to b} = \left(\sum_{i=1}^{n} |a_i - b_i|^2\right)^{1/2}$$

120

kNN Classification

- **How do you pick the best value for k?**
- Try a few values by pretending part of the training data is unknown, and identify which k value does the best job
- Low values (k=1, 2) can be noisy and subject to effect of outliers
- Large values for k smooth out the results, but can't be too big (so that categories with few datapoints get voted out)

121

k Nearest Neighbors – Pros & Cons

- **Pros:**
  - Simple to implement.
  - Training is trivial.
  - It can work with any number of classes.
  - Easy to add more data.
  - It has very few parameters (k and distance metric).
- **Cons:**
  - It has high computation cost.
  - It is not suitable for high dimensional data and categorical features.

122

kNN in action

```
>>> from sklearn.neighbors import KNeighborsClassifier

>>> knn = KNeighborsClassifier(n_neighbors=3)
>>> knn.fit(X_train, y_train)
>>> y_pred = knn.predict(X_test)

>>> from sklearn import metrics

>>> print("Accuracy:",metrics.accuracy_score(y_test, y_pred))

Accuracy: 0.9911421911421912
```

knn.ipynb

123