Advanced Data Analysis

DATA 71200

Class 6: Naive Bayes, Decision Trees,
SVMs, Uncertainty Estimates

Schedule

17-Jun	**Supervised Learning (Naive Bayes Classifiers and Decision Trees, Support Vector Machines, and Uncertainty estimates from Classifiers)**
18-Jun	Async: DataCamp
19-Jun	*No Class*
20-Jun	Async: DataCamp
24-Jun	Unsupervised Learning (Dimensionality Reduction & Feature Extraction, and Manifold Learning) Ethics (Part One) *Project 2 Due*

Project 2 (Due June 24)

Application of two supervised learning techniques on the dataset you created in Project 1. This assignment should be completed as a Jupyter notebook in your GitHub repository.

To **submit** the assignment, submit a link to your project Jupyter notebook on Blackboard.

The **goal** for this assignment is to apply different types of supervised learning algorithms with a range of parameter settings to your data and to observe which performs better.

Step 1: Load your data, including testing/training split from Project 1.

- Your testing and training split should be balanced
- Your data should be clean and missing data should be addressed

Step 2: (If not already done in Project 1) Prepare your data

- Make sure that all your appropriate variables are converted to categorical variables (as ordinal or one hot)
- Perform any necessary feature scaling

Step 3: Examine your target attribute. Based on the data exploration you did in Project 1, confirm and examine the attribute you are going to predict.

- Examine and plot the distribution of the target attribute in your training set (e.g., is it Gaussian, uniform, logarithmic). This will help you interpret the performance of different algorithms on your data.

Step 4: Selected two of the following supervised learning algorithms, ideally one from the first half of the list and one from the second half of the list

- K-Nearest Neighbor
- Linear Models
- Naïve Bayes
- Decision Trees
    - Single tree
    - Random Forest
    - Gradient Descent decision trees
- Support Vectors Machines

Step 5: For each of your selected models

- Run with the default parameters, training on your training set and testing on your testing set
    - Calculate precision, recall, and F1
- Run with the default parameters using cross-validation on the whole dataset
    - Calculate precision, recall, and F1
- (Where possible) adjust 2-3 parameters for each model using grid search
    - Report evaluation metrics for the best and worst-performing parameter settings

Tip: You should make notes on what worked well and what didn't. Such notes will be useful when you write up the paper for your final project.

Project 2 Rubric

	Missing	Fair	Good	Excellent
Data Prepatation	**0** (0.00%)	**2** (13.33333%) Missing saling and encoding, where appropriate, or data is inaccessible	**2.5** (16.66666%) Missing saling or encoding, where appropriate, or data is inaccessible	**3** (20.00%) Data is accessible, scaling and encoding done, where appropriate
Testing/Training	**0** (0.00%)	**0.5** (3.33333%) Not properly balanced or not used properly	**0** (0.00%)	**1** (6.66666%) Properly balanced and used
Target variable	**0** (0.00%)	**0.5** (3.33333%) Selected but not appropriate	**0** (0.00%)	**1** (6.66666%) Selected and appropriate
Algorithm 1- Default	**0** (0.00%)	**1.5** (10.00%) Run without cross-validation or appropriate evaluation metrics	**1.75** (11.66666%) Run without either cross-validation or appropriate evaluation metrics	**2** (13.33333%) Run with both cross-validation and appropriate evaluation metrics
Algorithm 1 - Parameters	**0** (0.00%)	**1.5** (10.00%) Run without 2 parameter adjustments and appropriate evaluation metrics	**1.75** (11.66666%) Run with either at least 2 parameter adjustments and appropriate evaluation metrics	**2** (13.33333%) Run with at least 2 parameter adjustments and appropriate evaluation metrics
Algorithm 2 - Default	**0** (0.00%)	**1.5** (10.00%) Run without cross-validation or appropriate evaluation metrics	**1.75** (11.66666%) Run with either cross-validation or appropriate evaluation metrics	**2** (13.33333%) Run with both cross-validation and appropriate evaluation metrics
Algorithm 2 - Parameters	**0** (0.00%)	**1.5** (10.00%) Run with neither 2 parameter adjustments and appropriate evaluation metrics	**1.75** (11.66666%) Run with either at least 2 parameter adjustments and appropriate evaluation metrics	**2** (13.33333%) Run with at least 2 parameter adjustments and appropriate evaluation metrics
Comments	**0** (0.00%)	**1.5** (10.00%) Present but not detailed enough	**1.75** (11.66666%) Largely complete and/or some detail missing	**2** (13.33333%) Complete with sufficient detail

Naive Bayes

$$P(A \mid B) = \frac{P(B \mid A) \cdot P(A)}{P(B)}$$

A, B	= events
$P(A\|B)$	= probability of A given B is true
$P(B\|A)$	= probability of B given A is true
$P(A), P(B)$	= the independent probabilities of A and B

- **Uses Bayes rule to predict the probability of an example belonging to a given class**
- **“Naive” because it examines each feature individually**
- **Calculate per-class statistics for each feature**
  - Bernoulli (binary data) - number of times a feature is non-zero for each class
  - Multinomial (count data) - average of how many times a feature occurs for each class
  - Gaussian (continuous data) - mean and standard deviation of the value of the feature for each class

Naive Bayes

Jupyter Notebook
data71200class6a.ipynb

- **Assumptions**
  - Independence between features
  - Gaussian version assumes normally distributed data with the same variance in each class

- **Best Practices**
  - Bernoulli and Multinomial best used on sparse data
  - Gaussian version best used on high-dimensional data

Naive Bayes

Jupyter Notebook
data71200class6a.ipynb

- **Parameters**
  - Alpha (Bernoulli and Multinomial) - model complexity
    - The larger the alpha, the more virtual data points that are added for smoothing
- **Strengths**
  - Faster to train and predict than linear models
  - Understandable training procedure
  - Works well on large datasets
    - More efficient on them than linear models
- **Weaknesses**
  - Coefficients not easily interpreted
  - Does not generalize as well as linear models

Decision Trees

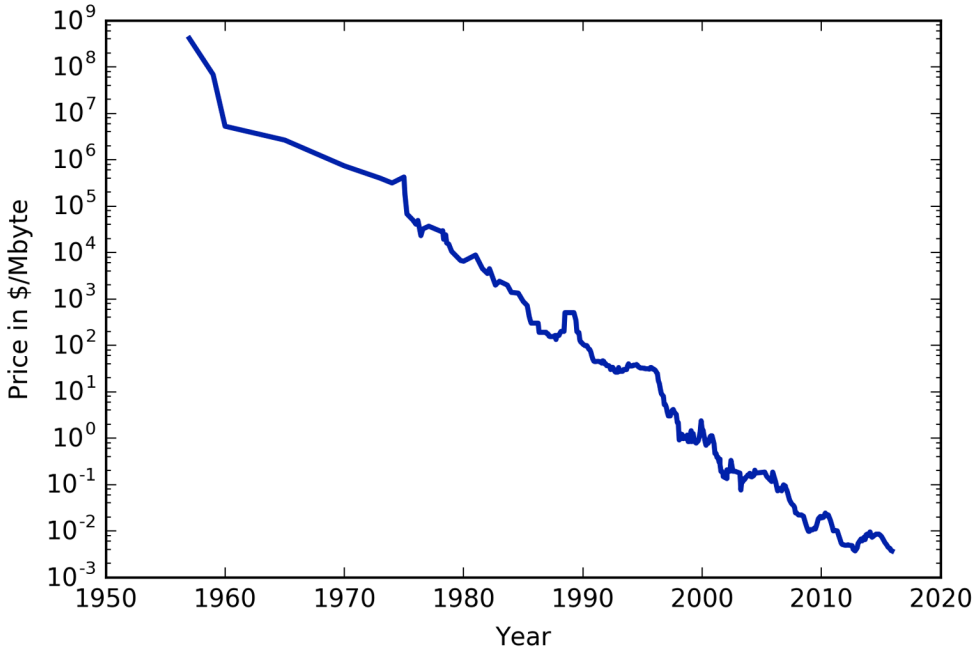
- Learned hierarchy of if/else questions
- Algorithm searches through all possible tests to separate the data into classes
- At each step the algorithm selects the test that provides the best separation of the classes
- Partitioning is repeated until each “leaf” contains only a single target (class or regression value) - referred to as pure

Guido, Sarah and Andreas C. Muller. (2016). Introduction to Machine Learning with Python, O’Reilly Media, Inc.

Decision Trees

- **Comprehensive tree building leads to overfitting the training data, this can be minimized by**
  - Pre-pruning - stopping the tree building early
    - Can be achieved by limiting the depth of the tree, number of leaves, or only splitting those with a certain number of points
  - Post-pruning/pruning - removing nodes that don’t contain much information
- **Visualizing the tree can provide information about how the algorithm makes decisions**
- **Feature importance is a summary of how important each feature is in the tree’s decision making**
  - Feature importance isn't class specific

Decision Trees Regression



- **Not able to make predictions outside of the range of the training data (extrapolate)**

Figure 2-31. Historical development of the price of RAM, plotted on a log scale

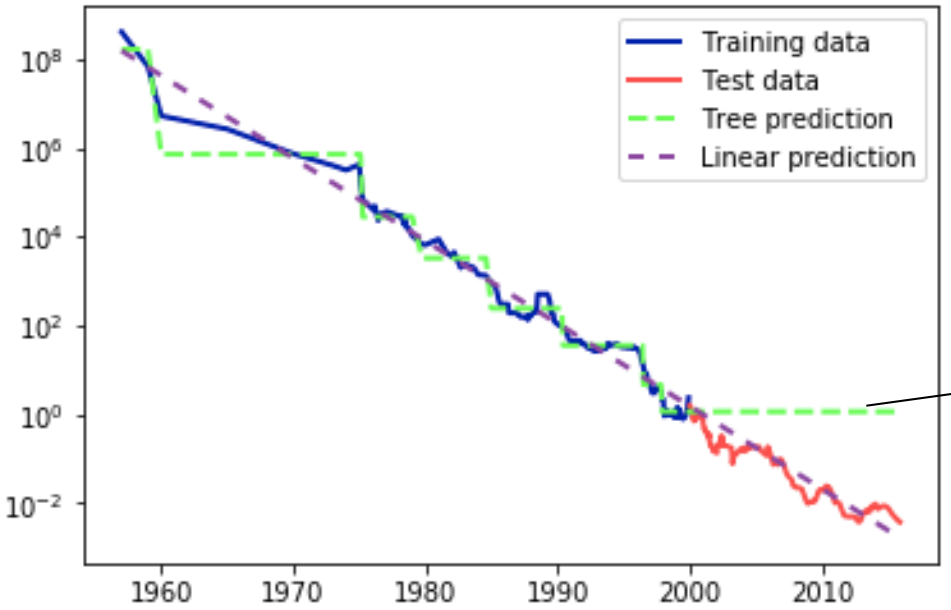


Figure 2-32. Comparison of predictions made by a linear model and predictions made by a regression tree on the RAM price data

Tree predicts the last known point

Guido, Sarah and Andreas C. Muller. (2016). Introduction to Machine Learning with Python, O'Reilly Media, Inc.

Decision Trees

Jupyter Notebook
data71200class6a.ipynb

- **Parameters**
  - Maximum depth - for pre-pruning
- **Assumptions**
  - No assumptions about the distribution of the data
- **Best Practices**
  - Use of pre-pruning prevents overfitting
- **Strengths**
  - Can be visualized, which aids in interpretation
  - Invariant to scaling data
  - Works well with mixed data (e.g., binary and continuous features)
- **Weaknesses**
  - Tend to overfit (even with pre-prunning) and thus don't always generalize
  - Doesn't work well on high-dimensional sparse data

Random Forests

Jupyter Notebook
data71200class6a.ipynb

- **Ensemble of slightly different decision trees**
  - Averaging the predictions of the trees minimizing overfitting
  - Difference between trees is achieved by randomizing which data points or features are used
- **Feature importance is aggregated across the ensemble of decision trees**
  - Typically more informative for random forests than for individual trees

Guido, Sarah and Andreas C. Muller. (2016). Introduction to Machine Learning with Python, O'Reilly Media, Inc.

Random Forests

Jupyter Notebook
data71200class6a.ipynb

- **Parameters**
  - Number of estimators - larger is always better
  - Number of data points - number of samples drawn from training dataset
  - Maximum features - amount of randomness (smaller reduces overfitting)
  - Maximum depth - for pre-prunning
- **Strength**
  - Invariant to scaling data
  - Works well with mixed data (e.g., binary and continuous features)
  - Generalizes better than decision trees
- **Weaknesses**
  - Harder to interpret than decision trees
  - Can be time consuming to train
  - Random process can make reproducibility difficult

Gradient Boosted Random Forests

- Series of decision trees where each new tree tries to correct the mistakes of the previous trees using pre-pruning (rather than randomness)
- Trees are typically shallow (weak learners)
- Feature importances tend to be sparser than random forests

Guido, Sarah and Andreas C. Muller. (2016). Introduction to Machine Learning with Python, O'Reilly Media, Inc.

Gradient Boosted Random Forests

Jupyter Notebook
data71200class6a.ipynb

- **Best Practices**
  - Random forests are a good place to start
    - Gradient-boosting can improve accuracy
- **Parameters**
  - Number of estimators - larger is always better
  - Maximum depth - for pre-prunning
  - Learning rate - how much a tree tries to correct the previous mistakes
- **Strengths**
  - Typically perform very well
  - Invariant to scaling data
  - Works well with mixed data (e.g., binary and continuous features)
- **Weaknesses**
  - Can take a long time to train
  - Doesn't work well on high-dimensional sparse data

Linear Support Vector Classifier

- **Find the linear classifier with the best separation (margin) between the two classes**
  - Operationally the data points used to make this calculation are the support vectors

Géron, Aurélien. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras and TensorFlow* O'Reilly Media, Inc.

Support Vector Machines

- Linear models are limited to separating classes with lines (hyperplanes in higher dimensions)
- Data can be transformed into a higher dimension
- Where an effective linear hyperplane can be fit to separate the data classes
- When projected back to the two original features (2D) the resultant decision boundary is not linear

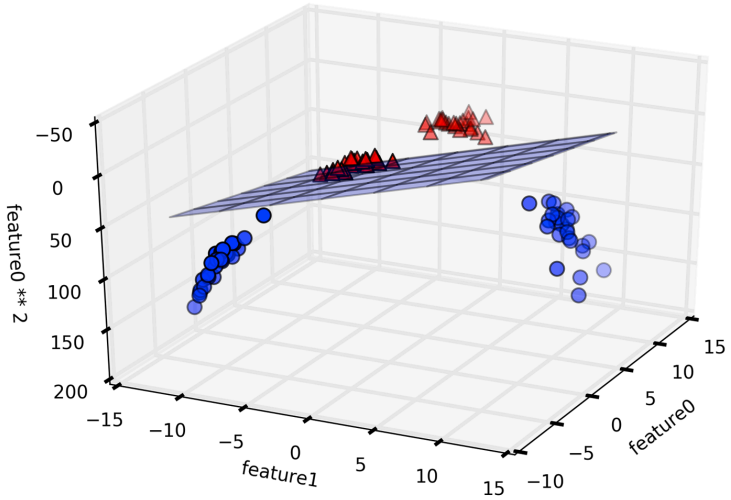


Figure 2-39. Decision boundary found by a linear SVM on the expanded three-dimensional dataset

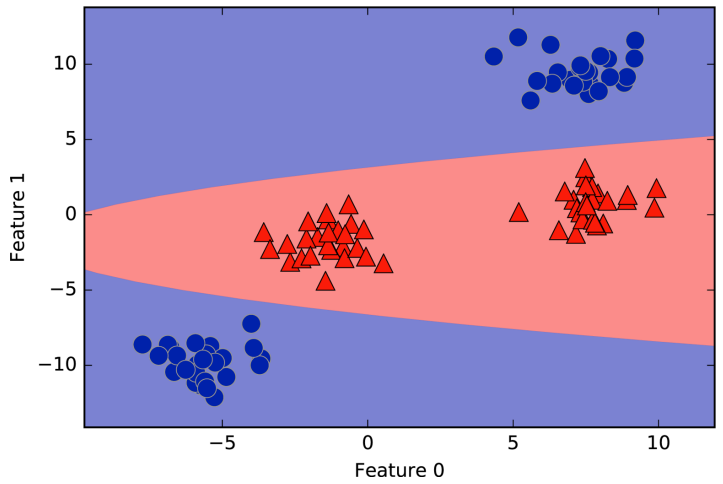


Figure 2-40. The decision boundary from Figure 2-39 as a function of the original two features

Guido, Sarah and Andreas C. Muller. (2016). Introduction to Machine Learning with Python, O'Reilly Media, Inc.

Support Vector Machines

- Adding nonlinear features to the data can be useful
- Kernel trick - computes the distances (scalar products) between the nonlinear features (rather than having to calculate the full representation)
  - Polynomial kernel - all possible polynomials up to a specified degree
  - Radial basis function (RBF) - Gaussian weighting of all possible polynomials

Guido, Sarah and Andreas C. Muller. (2016). Introduction to Machine Learning with Python, O'Reilly Media, Inc.

Support Vector Machines

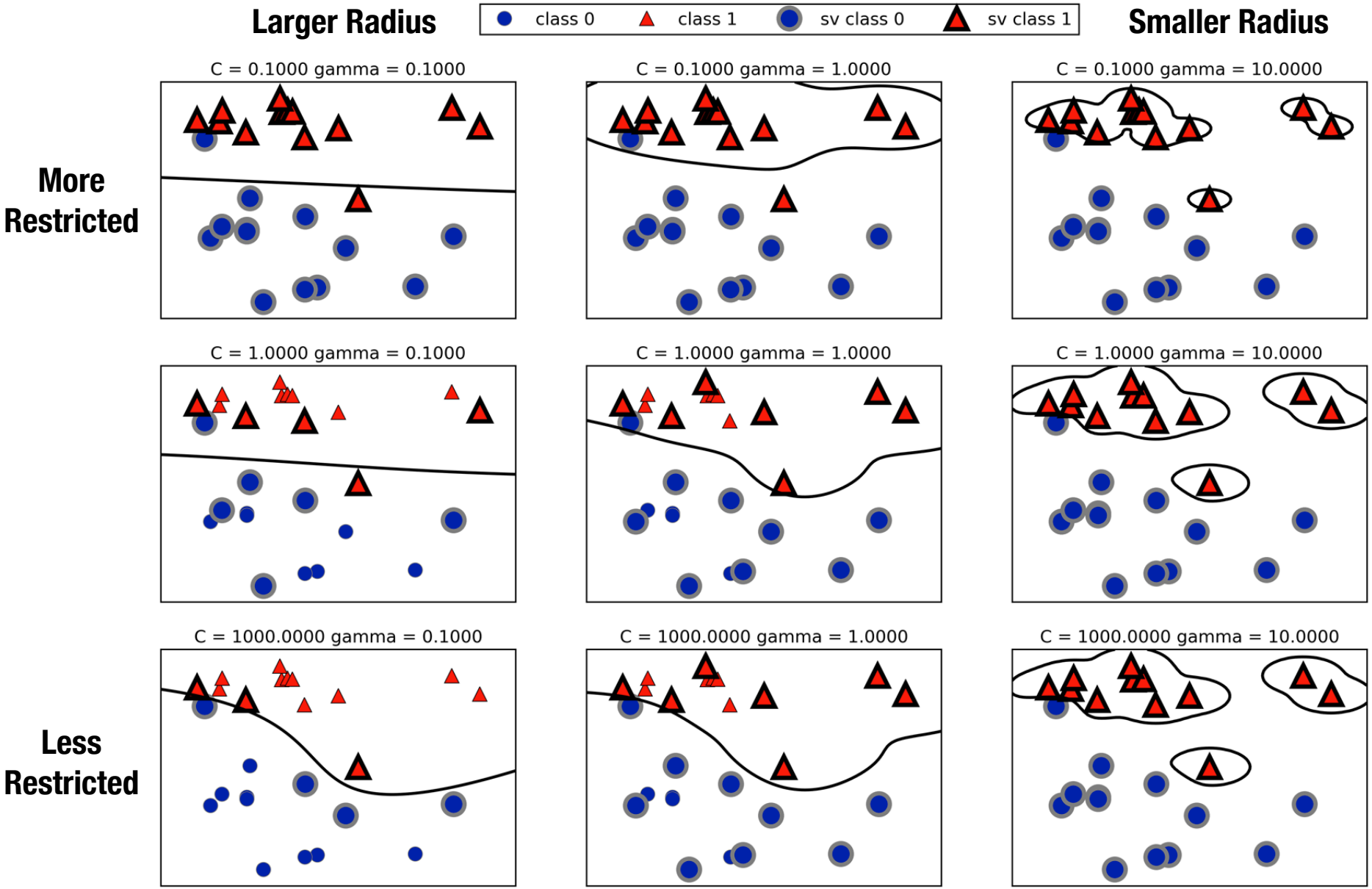


Figure 2-42. Decision boundaries and support vectors for different settings of the parameters C and gamma

Guido, Sarah and Andreas C. Muller. (2016). Introduction to Machine Learning with Python, O'Reilly Media, Inc.

Support Vector Machines

- **Best Practices**
  - Features should be scaled to have 0 mean and unit variance
- **Parameters**
  - Choice of kernel (polynomial, radial basis function, etc.)
  - C - regularization parameter
  - gamma - inverse of width of Gaussian kernel (RBF)
- **Strengths**
  - Works well with only a few data points
  - Works on both low- and high-dimensional data
- **Weaknesses**
  - Doesn’t scale very well to a large number of samples
  - Data often needs to be pre-processed

Uncertainty Estimates from Classifiers

- **Decision Functions encode how strongly the model believes a given datapoint belongs to a given class**

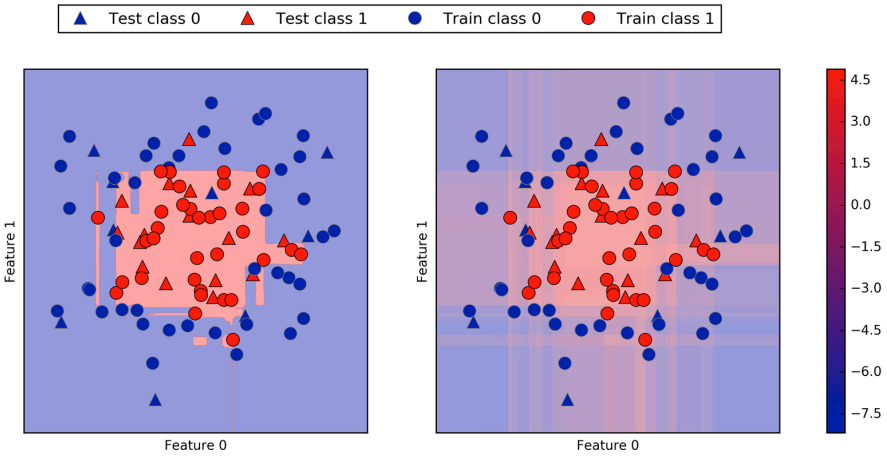


Figure 2-55. Decision boundary (left) and decision function (right) for a gradient boosting model on a two-dimensional toy dataset

Figure 2-56. Decision boundary (left) and predicted probabilities for the gradient boosting model shown in Figure 2-55

- **Prediction Probabilities indicate the probability (between 0 and 1) that the a given data point belongs to a given class**

Guido, Sarah and Andreas C. Muller. (2016). Introduction to Machine Learning with Python, O'Reilly Media, Inc.

Upcoming Work

- **DataCamp for June 18 and 20**
  - Machine Learning with Tree-Based Models in Python
  - Unsupervised Learning in Python
- **Videos for June 24 (password: data71200)**
  - Unsupervised Learning 1: https://vimeo.com/412723613
- **Reading for June 24**
  - Ch 3: “Unsupervised Learning” in Guido, Sarah and Andreas C. Muller. (2016). Introduction to Machine Learning with Python, O’Reilly Media, Inc. 133–170
  - Bostrom, Nick, and Eliezer Yudkowsky. (2014). “The ethics of artificial intelligence.” The Cambridge Handbook of Artificial Intelligence. 316–34.
- **Project 2 due on June 24**