Data Mining II
Regression
Regression

- **Classification**
  - covered in Data Mining I
  - predict a label from a finite collection
  - e.g., true/false, low/medium/high, ...

- **Regression**
  - predict a *numerical* value
  - from a possibly infinite set of possible values

- **Examples**
  - temperature
  - sales figures
  - stock market prices
  - ...
Contents

• A closer look at the problem
  – e.g., interpolation vs. extrapolation
  – measuring regression performance
• Revisiting classifiers we already know
  – which can also be used for regression
• Adoption of classifiers for regression
  – model trees
  – support vector machines
  – artificial neural networks
• Other methods of regression
  – linear regression
  – local linear regression
The Regression Problem

• Classification
  – algorithm “knows” all possible labels, e.g. yes/no, low/medium/high
  – all labels appear in the training data
  – the prediction is always one of those labels

• Regression
  – algorithm “knows” some possible values, e.g., 18°C and 21°C
  – prediction may also be a value not in the training data, e.g., 20°C
Interpolation vs. Extrapolation

- Training data:
  - weather observations for current day
  - e.g., temperature, wind speed, humidity, ...
  - target: temperature on the next day
  - target values between -15°C and 32°C

- Interpolating regression
  - only predicts values from the interval [-15°C, 32°C]

- Extrapolating regression
  - may also predict values outside of this interval
Interpolation vs. Extrapolation

- Interpolating regression is regarded as “safe”
  - i.e., only reasonable/realistic values are predicted

http://xkcd.com/605/
Interpolation vs. Extrapolation

- Sometimes, however, only extrapolation is interesting
  - how far will the sea level have risen by 2050?
  - will there be a nuclear meltdown in my power plant?

http://i1.ytimg.com/vi/FVfiujbGLfM/hqdefault.jpg
Baseline Prediction

- For classification: predict most frequent label
- For regression: predict average value
  - or median
  - or mode
  - in any case: only interpolating regression
- often a strong baseline

http://xkcd.com/937/
k Nearest Neighbors Revisited

- Problem
  - find out what the weather is in a certain place
  - where there is no weather station
  - how could you do that?
k Nearest Neighbors Revisited

- Idea: use the average of the nearest stations
- Example:
  - 3x sunny
  - 2x cloudy
  - result: sunny
- Approach is called
  - “k nearest neighbors”
  - where k is the number of neighbors to consider
  - in the example: k=5
  - in the example: “near” denotes geographical proximity
k Nearest Neighbors for Regression

- Idea: use the *numeric* average of the nearest stations
- Example:
  - 18°C, 20°C, 21°C, 22°C, 21°C
- Compute the average
  - again: k=5
  - \( (18+20+21+22+21)/5 \)
  - prediction: 20.4°C

- Only interpolating regression!
k Nearest Neighbor Regression in RapidMiner
Performance Measures

• Recap: measuring performance for classification:

\[ \text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \]

• If we use the numbers 0 and 1 for class labels, we can reformulate this as

\[ \sum_{\text{all examples}} |\text{predicted} - \text{actual}| \]

\[ \text{Accuracy} = 1 - \frac{\sum_{\text{all examples}} |\text{predicted} - \text{actual}|}{N} \]

Why?

– the nominator is the sum of all correctly classified examples
  • i.e., the difference of the prediction and the actual label is 0
– the denominator is the total number of examples
Mean Absolute Error

• We have

\[
\text{Accuracy} = 1 - \frac{\sum_{\text{all examples}} |\text{predicted} \ - \ \text{actual}|}{N}
\]

• For an arbitrary numerical target, we can define

\[
\text{MAE} = \frac{\sum_{\text{all examples}} |\text{predicted} \ - \ \text{actual}|}{N}
\]

• Mean Absolute Error
  – intuition: how much does the prediction differ from the actual value on average?
(Root) Mean Squared Error

- Mean Squared Error:
  \[ \text{MSE} = \frac{\sum_{\text{all examples}} |\text{predicted} - \text{actual}|^2}{N} \]

- Root Mean Squared Error:
  \[ \text{RMSE} = \sqrt{\frac{\sum_{\text{all examples}} |\text{predicted} - \text{actual}|^2}{N}} \]

- More severe errors are weighted higher by MSE and RMSE
Correlation

- Pearson's correlation coefficient
- Scores well if
  - high actual values get high predictions
  - low actual values get low predictions
- Caution: PCC is scale-invariant!
  - actual income: $1, $2, $3
  - predicted income: $1,000, $2,000, $3,000
    → PCC = 1

\[
PCC = \frac{\sum_{all\ examples} (\text{pred} - \overline{\text{pred}})(\text{act} - \overline{\text{act}})}{\sqrt{\sum_{all\ examples} (\text{pred} - \overline{\text{pred}})^2} \times \sqrt{\sum_{all\ examples} (\text{act} - \overline{\text{act}})^2}}
\]
Linear Regression

- Assumption: target variable $y$ is (approximately) linearly dependent on attributes
  - for visualization: one attribute $x$
  - in reality: $x_1...x_n$
Linear Regression

• Target: find a linear function $f$: $f(x) = w_0 + w_1x_1 + w_2x_2 + \ldots + w_nx_n$
  – so that the error is minimized
  – i.e., for all examples $(x_1, \ldots, x_n, y)$, $f(x)$ should be a *correct* prediction for $y$
  – given a performance measure
Linear Regression

• Typical performance measure used: Mean Squared Error
• Task: find $w_0,\ldots,w_n$ so that

$$\sum_{\text{all examples}} \left( w_0 + w_1 \cdot x_1 + w_2 \cdot x_2 + \ldots + w_n \cdot x_n - y \right)^2$$

is minimized

• note: we omit the denominator $N$
FIGURE 3.1. Linear least squares fitting with $X \in \mathbb{R}^2$. We seek the linear function of $X$ that minimizes the sum of squared residuals from $Y$. 
...but what about Non-linear Problems?
Possible Option: new Attributes

- The attributes $\mathbf{X}$ for linear regression can be:
  - Original attributes $\mathbf{X}$
  - Transformation of original attributes, e.g. log, exp, square root, square, etc.
  - Polynomial transformation
    - example: $y = \beta_0 + \beta_1 \cdot x + \beta_2 \cdot x^2 + \beta_3 \cdot x^3$
  - Basis expansions
  - Interactions between variables
    - example: $x_3 = x_1 \cdot x_2$
- This allows use of linear regression techniques to fit much more complicated non-linear datasets.
Example with Polynomially Transformed Attributes

\[ y(x, \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + \ldots + w_M x^M = \sum_{j=0}^{M} w_j x^j \]
Support Vector Machines Revisited

- Find hyperplane maximizes the margin => B1 is better than B2
Linear Regression and SVM

- **Linear Regression**
  - find a linear function that *minimizes* the distance to data points

- **Support Vector Machine**
  - find a linear function that *maximizes* the distance to data points (from different classes)

- **Both problems are similar**
  - hence, many SVMs also support regression
Support Vector Regression

• Maximum margin hyperplane only applies to classification
• However, idea of support vectors and kernel functions can be used for regression
• Basic method same as in linear regression: want to minimize error
  – Difference A: ignore errors smaller than $\varepsilon$ and use absolute error instead of squared error
  – Difference B: simultaneously aim to maximize flatness of function
• User-specified parameter $\varepsilon$ defines “tube”
Examples

\[ e = 2 \]

\[ e = 1 \]

\[ e = 0.5 \]
Local Regression

- Assumption: non-linear problems are approximately linear in local areas
  - idea: use linear regression locally
  - only for the data point at hand (lazy learning)
Local Regression

• A combination of
  – k nearest neighbors
  – local regression

• Given a data point
  – retrieve the k nearest neighbors
  – compute a linear regression model using those neighbors
  – locally weighted regression:
    uses distance as weight for error computation
Local Regression

• Advantage: fits non-linear models well
  – good local approximation
  – often more exact than pure k-NN

• Disadvantage
  – runtime
  – for each test example:
    • find k nearest neighbors
    • compute a local model
Combining Decision Trees and Regression

- Idea: split data first so that it becomes “more linear”
- example: fuel consumption by car weight
Combining Decision Trees and Regression

- Idea: split data first so that it becomes “more linear”
- example: fuel consumption by car weight
Combining Decision Trees and Regression

• Observation:
  – by cleverly splitting the data, we get more accurate linear models

• Regression trees:
  – decision tree for splitting data
  – constants as leaves

• Model trees:
  – more advanced
  – linear functions as leaves

\[
y = 0.005x + 1 \quad \text{if fuel type} = \text{diesel}
\]

\[
y = 0.01x + 2 \quad \text{if fuel type} = \text{benzine}
\]
Regression Trees

- Like decision trees, but:
  - Splitting criterion: minimize intra-subset variation
  - Termination criterion: std dev becomes small
  - Pruning criterion: based on numeric error measure
  - Prediction: Leaf predicts average class values of instances

- Easy to interpret
- More sophisticated version: *model trees*
- Resulting model: piecewise constant function
Model Trees

• Build a regression tree
• Each leaf $\Rightarrow$ linear regression function
• Need linear regression function at each node
• At each node, use only a subset of attributes
  – Those occurring in subtree
  – (+maybe those occurring in path to the root)
• Fast: tree usually uses only a small subset of the attributes
• Prediction: go down tree, then apply function
• Resulting model: piecewise linear function
Local Regression

• Assumption: non-linear problems are approximately linear in local areas
  – idea: use linear regression locally
  – only for the data point at hand (lazy learning)
Building the Tree

• Splitting: standard deviation reduction
  \[ SDR = sd(T) - \sum_i \left| \frac{T_i}{T} \right| \times sd(T_i) \]

• Termination:
  – Standard deviation < 5% of its value on full training set
  – Too few instances remain (e.g. < 4)

• Pruning:
  – Heuristic estimate of absolute error of LR models:
    \[ \frac{n+v}{n-v} \times \text{average\_absolute\_error} \]
    Greedily remove terms from LR models to minimize estimated error
  – Heavy pruning: single model may replace whole subtree
  – Proceed bottom up: compare error of LR model at internal node to error of subtree
Missing Values

- Modify splitting criterion:
  \[
  SDR = \frac{m}{|T|} \times [sd(T) - \sum_i \frac{T_i}{T} \times sd(T_i)]
  \]

- To determine which subset an instance goes into, use *surrogate splitting*
  - Split on the attribute whose correlation with original is greatest
  - Problem: complex and time-consuming
  - Simple solution: always use the class

- Test set: replace missing value with average
Surrogate Splitting Based on Class

- Choose split point based on instances with known values
- Split point divides instances into 2 subsets
  - $L$ (smaller class average)
  - $R$ (larger)
- $m$ is the average of the two averages
- For an instance with a missing value:
  - Choose $L$ if class value $< m$
  - Otherwise $R$
- Once full tree is built, replace missing values with averages of corresponding leaf nodes
Pseudo-code for M5'

• Four methods:
  – Main method: MakeModelTree
  – Method for splitting: split
  – Method for pruning: prune
  – Method that computes error: subtreeError

• We’ll briefly look at each method in turn

• Assume that linear regression method performs attribute subset selection based on error
MakeModelTree

MakeModelTree (instances)
{
    SD = sd(instances)
    for each k-valued nominal attribute
        convert into k-1 synthetic binary attributes
    root = newNode
    root.instances = instances
    split(root)
    prune(root)
    printTree(root)
}
split

split(node)
{
    if sizeof(node.instances) < 4 or
        sd(node.instances) < 0.05*SD
        node.type = LEAF
    else
        node.type = INTERIOR
    for each attribute
        for all possible split positions of attribute
            calculate the attribute's SDR
        node.attribute = attribute with maximum SDR
    split(node.left)
    split(node.right)
}
prune

prune(node)
{
    if node = INTERIOR then
        prune(node.leftChild)
        prune(node.rightChild)
        node.model = linearRegression(node)
        if subtreeError(node) > error(node) then
            node.type = LEAF
}
subtreeError

subtreeError(node)
{
    l = node.left; r = node.right
    if node = INTERIOR then
        return (sizeof(l.instances)*subtreeError(l) + sizeof(r.instances)*subtreeError(r)) /sizeof(node.instances)
    else return error(node)
}
Rules from Model Trees

• PART algorithm generates classification rules by building partial decision trees
• Can use the same method to build rule sets for regression
  – Use model trees instead of decision trees
  – Use variance instead of entropy to choose node to expand when building partial tree
• Rules will have linear models on right-hand side
Output $Y$ is 1 if at least two of the three inputs are equal to 1.
Artificial Neural Networks Revisited

\[ Y = I(0.3X_1 + 0.3X_2 + 0.3X_3 - 0.4 > 0) \]

where \( I(z) = \begin{cases} 1 & \text{if } z \text{ is true} \\ 0 & \text{otherwise} \end{cases} \)
Artificial Neural Networks Revisited

\[ Y = I \left( 0.3X_1 + 0.3X_2 + 0.3X_3 - 0.4 > 0 \right) \]

where \( I(z) = \begin{cases} 1 & \text{if } z \text{ is true} \\ 0 & \text{otherwise} \end{cases} \)
Artificial Neural Networks Revisited

• This final function was used to separate two classes:
  \[ Y = I \left( 0.3X_1 + 0.3X_2 + 0.3X_3 - 0.4 > 0 \right) \]

  where \( I(z) = \begin{cases} 1 & \text{if } z \text{ is true} \\ 0 & \text{otherwise} \end{cases} \)

• However, we may simply use it to predict a numerical score (between 0 and 1) by changing it to:
  \[ Y = 0.3X_1 + 0.3X_2 + 0.3X_3 - 0.4 \]
Artificial Neural Networks for Regression

• What has changed:
  – we do not use a cutoff for 0/1 predictions
  – but leave the numbers as they are

• Training examples:
  – attribute vectors – not with a class label, but numerical target
Artificial Neural Networks for Regression

\[ Y = 0.3X_1 + 0.3X_2 + 0.3X_3 - 0.4 \]
Artificial Neural Networks for Regression

• Given that our target formula is of the form

\[ Y = 0.3X_1 + 0.3X_2 + 0.3X_3 - 0.4 \]

• we can learn only linear problems
  – i.e., the target variable is a linear combination the input variables

• More complex regression problems can be approximated
  – by combining several perceptrons
  – this allows for arbitrary polynomial functions (degree: number of layers)

• In principle, ANNs also do extrapolation
  – although they mostly behave interpolating
Artificial Neural Networks Revisited

Training ANN means learning the weights of the neurons
Summary

• Regression
  – predict numerical values instead of classes

• Performance measuring
  – absolute or relative error, correlation, …

• Methods
  – k nearest neighbors
  – linear regression
  – SVMs
  – model trees
  – artificial neural networks
Questions?