Algorithms to Construct Models
  Decision Trees
  kNN
Formulae
  Entropy/Information
  Evaluation
    Accuracy
    Error
    Precision
    Recall
  Bias-Variance Decomposition
    Expected Error
    Bias
    Variance
  VC Dimensions
Characteristics of Decision Boundaries of Each ML Algorithm & Each Kernels
Problem Setting of Regression Models
Parameters
  in SVM

Appendix

Algorithms to Construct Models
<table>
<thead>
<tr>
<th></th>
<th>Deci.Tree.</th>
<th>kNN</th>
<th>Poly.reg.</th>
<th>Log.reg.</th>
<th>SVM</th>
<th>Perceptron</th>
</tr>
</thead>
<tbody>
<tr>
<td>Meant For...</td>
<td>Classific.</td>
<td>Class.</td>
<td>Regress.</td>
<td>Class.</td>
<td>Class.</td>
<td>Class.</td>
</tr>
<tr>
<td>Param. To Prevent Overfit</td>
<td>d depth</td>
<td>k — # of NNs to see</td>
<td>d degree</td>
<td>&quot;regulation&quot; λ on weight</td>
<td>&quot;Tradeoff&quot; C on training data</td>
<td>-</td>
</tr>
<tr>
<td>Type</td>
<td>Rule-Based Learning</td>
<td>Instance-Based Learning</td>
<td>Regression Analysis</td>
<td>Regression Analysis</td>
<td>IBL &gt; Kernel Method</td>
<td>0-1 Binary Classifier</td>
</tr>
<tr>
<td>Simplest Form</td>
<td>d=1 : Deci.Stump</td>
<td>k=1 : 1NN</td>
<td>d=1 : lin.reg.</td>
<td>-</td>
<td>Linear SVM</td>
<td>-</td>
</tr>
<tr>
<td>Optimization Algorithm</td>
<td>Build Tree</td>
<td>Find k-th NN</td>
<td>Gradient Descent</td>
<td>Gradient Descent</td>
<td>Gradient Descent</td>
<td>Gradient Descent</td>
</tr>
<tr>
<td>Standardization Needed? / Model Is Scale Variant?</td>
<td>×</td>
<td>×</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>×</td>
</tr>
<tr>
<td>Feature Mapping?</td>
<td>Perhaps Helpful</td>
<td>Not Useful</td>
<td>Perhaps Helpful</td>
<td>Perhaps Helpful</td>
<td>Required, Built-In and Core: Kernel</td>
<td>Perhaps Helpful</td>
</tr>
<tr>
<td>Supervised?</td>
<td>×</td>
<td>×</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>Decision Boundary Looks Like...</td>
<td>Stairs. Axis-parallel.</td>
<td>Voronoi cells.</td>
<td>Curve of degree d</td>
<td>Linear</td>
<td>Depends on Kernel</td>
<td>Linear (hyperplane)</td>
</tr>
<tr>
<td>Online/Batch Learning</td>
<td>Batch</td>
<td>Batch</td>
<td>Batch / Online</td>
<td>Batch / Online</td>
<td>Batch / Online</td>
<td>Batch / Online</td>
</tr>
<tr>
<td>Loss Function</td>
<td>0-1 Loss (No Training!)</td>
<td>sum of squared error</td>
<td>Log Loss (&quot;Sigmoid&quot;)</td>
<td>Hinge Loss</td>
<td>Hinge Loss</td>
<td></td>
</tr>
<tr>
<td>Sub-Loss Func.</td>
<td>InfoGain / SplitInfo</td>
<td>Euclidean Distance</td>
<td>$y_i - h_\theta(x_i)$</td>
<td>$y_i - h_\theta(x_i)$</td>
<td>$y_i - h_\theta(x_i)$</td>
<td>$y_i \cdot h_\theta(x_i)$</td>
</tr>
<tr>
<td>Hypothesis Function (if any) $h_\theta(x_i) = \beta^T x_i$</td>
<td>-</td>
<td>-</td>
<td>$\beta^T x_i$</td>
<td>$g(\beta^T x_i)$</td>
<td>$\beta^T x_i$</td>
<td>$\beta^T x_i$</td>
</tr>
</tbody>
</table>

### Decision Trees

```python
def create_subtree:
    if algorithm == "ID3" : calculate_score = calculate_infoGain
    elif algorithm == "C4.5" : calculate_score = calculate_infoGain /
calculate_splitInfo
# Main:
scores = {attribute: calculate_score(attribute,
                        attribute.all_possible_values)
        for attribute in all attributes}
best_attribute = score.the_attribute_with_highest_score
return (best_attribute, {value: create_subtree(where best_attribute ==
value)
        for value in best_attribute.all_possible_values})
```

### kNN
**Formulae**

**Entropy/Information**

\[ H(X) = - \sum_{i=1}^{n} P(X_i) \log_2 P(X_i) \]

**Evaluation**

**Accuracy**

\[ \text{accuracy} = \frac{\# \text{ of correct predictions}}{\# \text{ of test examples}} \]

**Error**

\[ \text{error} = 1 - \text{accuracy} = \frac{\# \text{ of incorrect predictions}}{\# \text{ of test examples}} \]

**Precision**

\[ \text{precision} = \frac{\# \text{ of test examples predicted to be } + \text{ and labeled as } +}{\# \text{ of test examples predicted to be } +} \]

**Recall**

\[ \text{recall} = \frac{\# \text{ of test examples predicted to be } + \text{ and labeled as } +}{\# \text{ of test examples labeled to be } +} \]

**Bias-Variance Decomposition**

**Expected Error**

\[ E\left[ (y - f(x))^2 \right] = \text{Bias}_f^2 + \text{Variance}_f + \text{Noise} \]

**Bias**

\[ \text{Bias}[\hat{f}(x)] = E[\hat{f}(x) - f(x)] \]

The error caused by the simplifying assumptions built into the method. / The error caused by using a simpler model to approximate data w/ a more complex trend.

- **Low Bias**: Suggests less assumptions about the form of the target function.
- **High-Bias**: Suggests more assumptions about the form of the target function.
Variance

How much the model will move around its mean if we provided different set of training data.

- **Low Variance**: Suggests small changes to the estimate of the target function with changes to the training dataset.
- **High Variance**: Suggests large changes to the estimate of the target function with changes to the training dataset.

VC Dimensions

If you can find a set of \( n \) points, so that it can be shattered by the classifier (i.e. classify all possible \( 2^n \) labelings correctly) and you cannot find any set of \( n + 1 \) points that can be shattered (i.e. for any set of \( n + 1 \) points there is at least one labeling order so that the classifier can not separate all points correctly), then the VC dimension is \( n \).

Example: A line can shatter 3 points.

![Decision Boundaries](image)

Characteristics of Decision Boundaries of Each ML Algorithm & Each Kernels

- **Random Forest & AdaBoost w/ weak hypothesis == decision boundary**: Much alike, but Adaboost leaves certain blocks in the hypothesis space unable to be determined.
- **Logistic Regression & Linear Regression & Linear SVM**: Gives linear decision boundaries.
- **Decision Tree**: Stairs. Axis-parallel.
Nearest Neighbor: Voronoi cells.

Problem Setting of Regression Models

1. Load raw data file.
2. (Optional) Make more features using `mapFeatures()`.
4. Separately Standardize two datasets.
5. Input — $d$ features of $n$ training examples: $X$.
6. Prepend a column of 1's to $X$.
7. Apply our model $h_{\theta}(x)$ — A model is what maps an example $x$ to a label $y$ (this process is called prediction). (This function itself is called the activation function of this model.)
o Linear Regression uses **Linear Model**: \( h_\theta(x) = \theta^T x \).

o Logistic Regression uses **Logistic Model**: \( h_\theta(x) = \frac{1}{1+e^{-\theta^T x}} \).
  - The **Logistic / Sigmoid Function** wraps over, and “replaces”, the "error function" \( h_\theta(x) \).
  - Perceptron: \( h_\theta(x) = \text{sign}(\theta^T x) \)

8. Use **gradient descent** — how much our \( \theta \) have to change, in order to achieve lower cost.

1. Calculate the **gradient** of the **cost function** w.r.t. features \( j = 1, \ldots, d + 1 \):
   - For **Linear Reg.**:
     \[
     \nabla = \frac{1}{n} \sum_{i=1}^n [h_\theta(x_i) - y_i] \cdot x_j
     \]
   - For **Logreg**:
     \[
     \nabla = \sum_{i=1}^n [h_\theta(x_i) - y_i] \cdot x_j
     \]

BTW, it’s the derivative of the **objective function** \( J(\theta) \), sum of **cost functions** (errors) in training:

   - For **Linear Reg.**, squared errors: \( J(\theta) = \sum_{i=1}^n \frac{1}{2n} (h_\theta(x_i) - y_i)^2 \).

2. Add step control \( \alpha \), and optionally add regularization \( \lambda \):

   - For **Logreg**, **individual error weighted with \( x_i \)**:
     \[
     J(\theta) = -\sum_{i=1}^n [y_i \log h_\theta(x_i) + (1-y_i) \cdot \log(1-h_\theta(x_i))] \cdot x_i
     \]
   - For **Perceptron** (under <u>Batch Learning</u>):
     \[
     J(\theta) = \frac{1}{n} \sum_{i=1}^n \max(0, -y_i \cdot \theta^T x_i)
     \]

3. Update model parameters \( \theta \) with the grad.: \( \theta \leftarrow \theta - \nabla \)

   - **Perceptron Rule**:
     - **Online Learning**: \( \theta \leftarrow \theta + y_i \cdot x_i \) only upon misclassification.
     - **Batch Learning**: \( \theta \leftarrow \theta + \alpha \cdot \Delta \), where \( \Delta = \sum y_i \cdot x_i \) that are misclassified.

4. Repeat from Step 1 till convergence (or max step count exceeded).

5. To use the model, we simply calculate \( h_\theta(x) \). Again,

   - Linear Regression uses **Linear Model**: \( h_\theta(x) = \theta^T x \).
   - Logistic Regression uses **Logistic Model**: \( h_\theta(x) = \frac{1}{1+e^{-\theta^T x}} \).

   - Remember that the motivation of inventing Logreg is to get classifications instead of predictions (like linear reg gives). Therefore, a \( \text{round()} \) is needed.
This is NOT to say that we cannot use linreg for prediction; it's just not meant for that.

## Parameters

### C in SVM

The C parameter tells the SVM optimization how much you want to avoid misclassifying each training example. For large values of C, the optimization will choose a smaller-margin hyperplane if that hyperplane does a better job of getting all the training points classified correctly. Conversely, a very small value of C will cause the optimizer to look for a larger-margin separating hyperplane, even if that hyperplane misclassifies more points. For very tiny values of C, you should get misclassified examples, often even if your training data is linearly separable.

The SVM has low bias and high variance, but the trade-off can be changed by increasing the C parameter that influences the number of violations of the margin allowed in the training data which increases the bias but decreases the variance.

### λ

Regularization factor. Found in $\nabla \equiv \alpha \{ \sum_{i=1}^{n} [h_\theta(x_i) - y_i] \cdot x_i + \lambda \theta_j \} (\text{but no } \lambda \text{ if } j = 1)$.

Increasing $\lambda$, we can reduce variance but increase bias.

The regularization parameter $\lambda$ is a control on your fitting parameters. As the magnitudes of the fitting parameters increase, there will be an increasing penalty on the cost function. This penalty is dependent on the squares of the parameters as well as the magnitude of $\lambda$. Also, notice that the summation after $\lambda$ does not include $\theta_0^2$.

Visually, increasing $\lambda$, see this.

### k

Number of neighbors to consider.

Used in kNN classifiers.

## Appendix

- Too little or too much training data could both cause overfitting.
1. Perceptron can be considered as a Linear SVM w/o margin (result-wise). Compared to Linreg: [here]({#})

2. Of course they may be used for the other purpose too, just not so smoothly. [↩](#)

3. and those in the kernel, if any. e.g.: $\lambda$ [↩](#)

4. except un-regularized linear regression with closed form solution [↩](#)

5. Not really “decision boundary”! [↩](#)

6. For each training example, let the tree predict. If the prediction is wrong, branch this leaf (1); if right, we do nothing (0). [↩](#)

7. In Regression, that's the $h_\theta$ — "hypothesis function with the current values of theta". [↩](#)