Algorithms to Construct Models

	Deci.Tree.	**kNN**	**Poly.reg.**	**Log.reg.**	**SVM**	**Perceptron** [1]
Meant For... [2]	Classifica.	Class.	Regress.	Class.	Class.	Class.
Param. To Prevent Overfit	`d` depth	`k` — # of NNs to see	`d` degree	"regulation" λ on **weight**	"Tradeoff" C on **training data** [3]	-
Type	Rule-Based Learning	Instance-Based Learning	Regression Analysis	Regression Analysis	IBL > Kernel Method	0-1 Binary Classifier
Simplest Form	`d=1` : Deci.Stump	`k=1` : 1NN	`d=1` : lin.reg.	-	Linear SVM	-
Optimization Algorithm	Build Tree	Find k-th NN	Gradient Descent	Gradient Descent	Gradient Descent	Gradient Descent
Standardization Needed? / Model Is Scale Variant?	×	×	√ [4]	√	√	×
Feature Mapping?	Perhaps Helpful	Not Useful	Perhaps Helpful	Perhaps Helpful	Required, Built-In and Core: **Kernel**	Perhaps Helpful
Supervised?	×	×	√	√	√	√
Decision Boundary Looks Like...	Stairs. Axis-parallel.	Voronoi cells.	Curve of degree `d` [5]	Linear	Depends on Kernel	Linear (hyperplane)
Online/Batch Learning	Batch	Batch	Batch / Online	Batch / Online	Batch / Online	Batch / Online
Loss Function	0-1 Loss [6]	(No Training!)	sum of squared error	Log Loss ("Sigmoid")	Hinge Loss	Hinge Loss
Sub-Loss Func.	InfoGain / SplitInfo	Euclidean Distance	$y_i - h_\theta(x_i)$	$y_i - h_\theta(x_i)$	$y_i - h_\theta(x_i)$	$y_i \cdot h_\theta(x_i)$
Hypothesis Function (if any) $h_\theta(x_i) =$ [7]	-	-	$\theta^{\mathrm{T}} x_i$	$g(\theta^{\mathrm{T}} x_i)$	$\theta^{\mathrm{T}} x_i$	$\theta^{\mathrm{T}} x_i$

Decision Trees

```
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

```
return the label of the k-th nearest neighbor
```

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

$$\text{precision} = \frac{\text{\# of test examples predicted to be \& labeled as } +}{\text{\# of test examples predicted to be } +}$$

Recall

$$\text{recall} = \frac{\text{\# of test examples predicted to be \& labeled as } +}{\text{\# of test examples labeled to be } +}$$

Bias-Variance Decomposition

Expected Error

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

Bias

$$\text{Bias}\left[\hat{f}\,(x)\right] = \text{E}\left[\hat{f}\,(x) - f(x)\right]$$

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

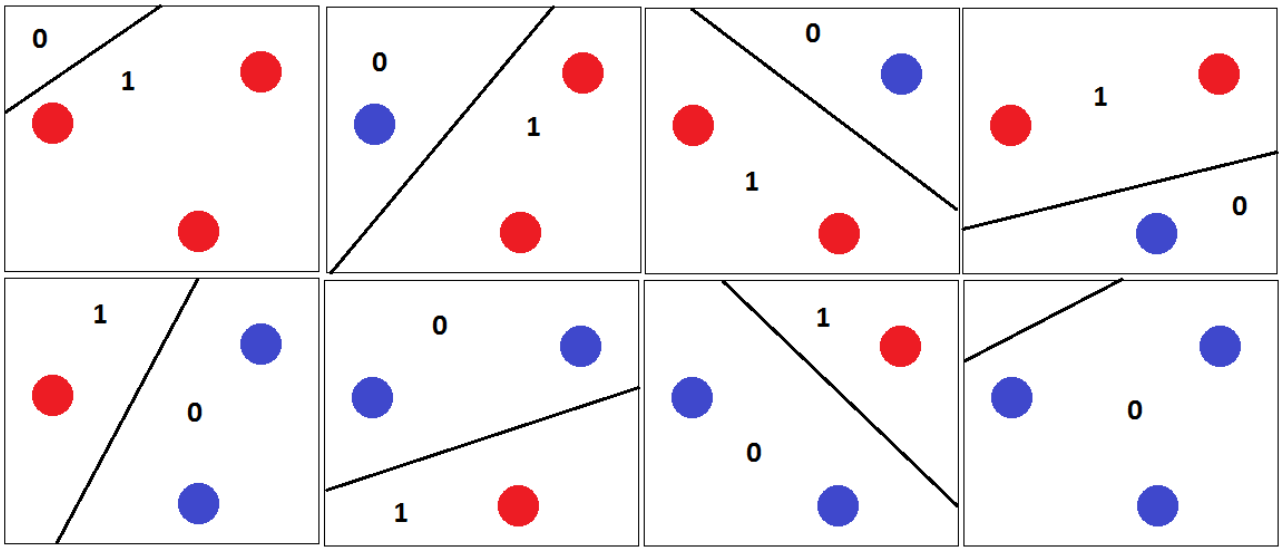
How much the model will move around its mean if we provided different set of training data.

- **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 $2n$ 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 seperate all points correctly), then the VC dimension is n.

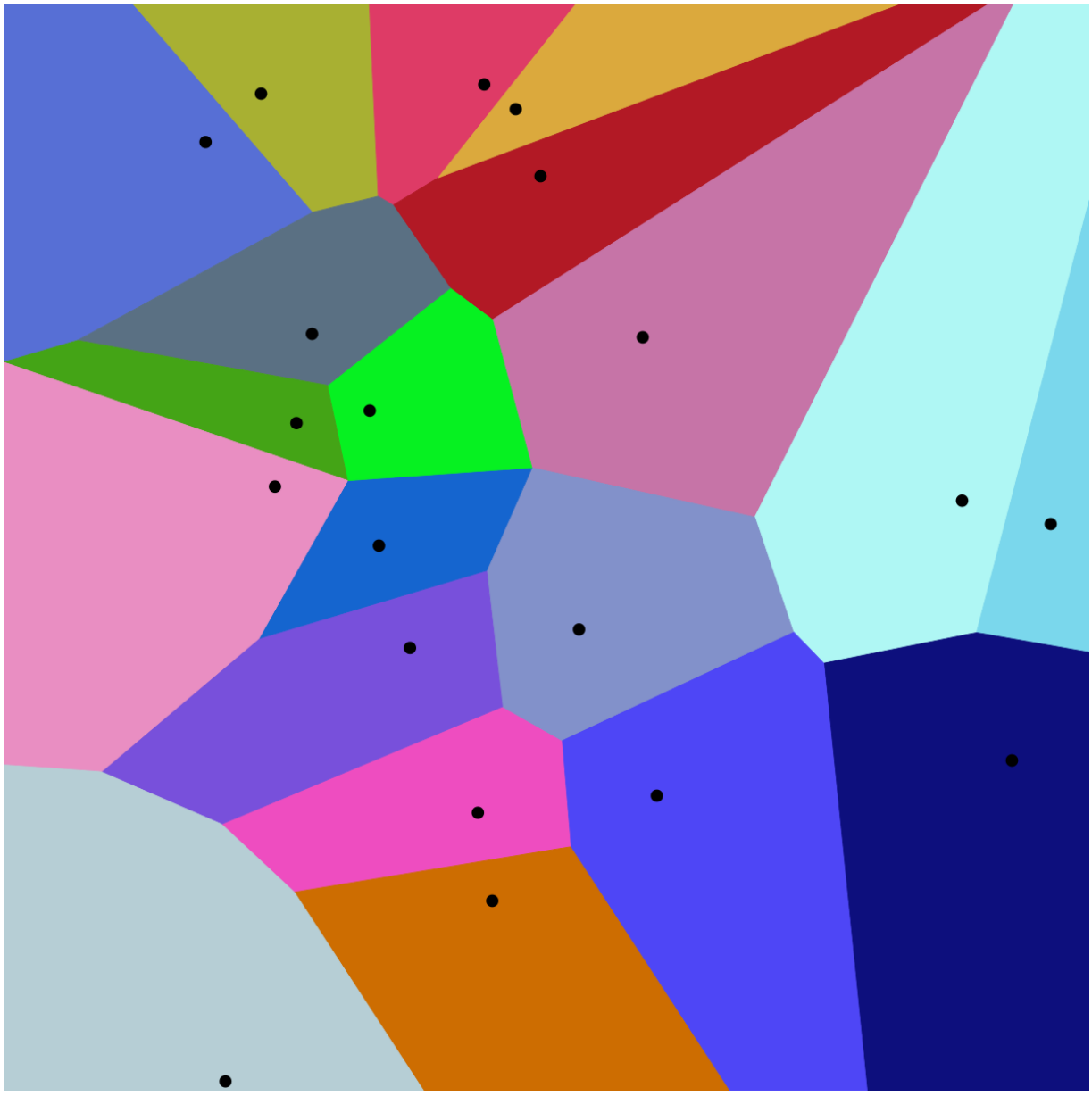
Example: A line can shatter 3 points.



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.
- **Logisitic 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()`.
3. Split data into training set and test set.
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 *perdiction*). *(This function itself is called the* ***activation function*** *of this model.)*

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

8. Use **gradient descent** — how much our θ 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.**: $\frac{1}{n}\sum_{i=1}^{n}[h_\theta(x_i) - y_i] \cdot x_j$
      - For **Logreg**: $\sum_{i=1}^{n}[h_\theta(x_i) - y_i] \cdot x_j$

      BTW, it's the derivative of the **objective function** $J(\theta)$, sum of **cost functions** (errors) in training:
      - For **Linear Reg.**, squared errors: $J(\theta) = \sum_{i=1}^{n} \frac{1}{2n}(h_\theta(x_i) - y_i)^2$.

```
  - For **Logreg**, <u>individual error weighted with $x_i$</u>: $J(\theta)=-
\sum_{i=1}^n[y_i\log h_\theta (x_i)+(1-y_i)\cdot \log(1-h_\theta (x_i))]$.
  - For **Perceptron** (under <u>Batch Learning</u>): $J(\theta)=\frac1n
\sum_{i=1}^n\max(0,-y_i \theta^\text{T}x_i)$.

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

  - For **Linear Reg**.: $\nabla\equiv$ ==$\alpha$==$\frac1{n}\sum_{i=1}^n
[h_\theta (x_i)-y_i]\cdot x_j$
```

```
  - For **Logreg**:          $\nabla\equiv$ ==$\alpha$==  $\{\sum_{i=1}^n
[h_\theta (x_i)-y_i]\cdot x_j$==$+\lambda\theta_j$==$\}$ (but no $\lambda$ if
$j=1$)
```

3. Update model parameters θ 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**: $h_\theta(x) = \theta^{\mathrm{T}} x$.
   - Logistic Regression uses **Logistic Model**: $h_\theta(x) = \frac{1}{1+e^{-\theta^{\mathrm{T}} x}}$.
     - Remember that the motivation of inventing Logreg is to get classifications instead of predictions (like linear reg gives). Therefore, a `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.[8]

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

Increasing λ, we can reduce variance but increase bias.

The regularization parameter λ is a control on your fitting parameters. As the magnitues 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 λ.** Also, notice that the summation after λ does not include θ_0^2.

Visually, increasing λ, 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.: λ ↩

4. except un-regularized linear regression with closed form solution↩

5. Not really "decision boundary"!↩

6. For each training example, let the tree perdict. If the perdiction is wrong, branch this leaf (1); if right, we do nothing (0).↩

7. In Regression, that's the h_θ — "hypothesis function with the current values of theta". ↩

8. Source: https://stats.stackexchange.com/questions/31066/what-is-the-influence-of-c-in-svms-with-linear-kernel↩