Section 3: Final Review

STATS 202: Statistical Learning and Data Science

Linh Tran

Department of Statistics
Stanford University

August 8, 2025

Announcements

- Last recording of the quarter.
- Kaggle predictions due Sunday night.
- Final project write-up is due Wednesday.
  - **Reference your Kaggle leaderboard name on Page 1**
- Final exam is next Saturday
  - Time: Saturday August 16 @ 7:00 PM -10:00 PM
  - Location: 300-300
  - 8 questions (lowest question dropped)
  - Practice exam to be released tonight (solutions next week)
  - Accommodation requests should already be made
- Course evaluation starts on Aug 11 (on Canvas).

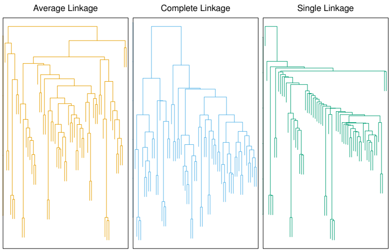
Outline

- Course review

Unsupervised learning

- In unsupervised learning, all the variables are on equal standing, no such thing as an input and response.
- Clustering is typically applied
  - Hierarchical clustering (single, complete, or average linkage).
  - K-means clustering.
  - Expectation maximization (using Gaussian mixtures).

Hierarchical clustering



- Agglomerative algorithm produces a *dendrogram*.
- At each step we join the two clusters that are "closest":
  - **Complete:** distance between clusters is maximal distance between any pair of points.
  - **Single:** distance between clusters is minimal distance.
  - **Average:** distance between clusters is the average distance.
- Height of a branching point = distance between clusters joined.

K-means clustering

- The number of clusters is fixed at K.
- Goal is to minimize the average distance of a point to the average of its cluster.
- The algorithm starts from some assignment, and is guaranteed to decrease this average distance.
- This find a local minimum, not necessarily a global minimum, so we typically repeat the algorithm from many different random starting points.

Supervised learning

We're interested in a response variable Y associated to each vector of predictors $\mathbf{X}$.

Regression: $f_0 = \mathbb{E}_0[Y|X_1, X_2, ..., X_p]$

- A scalar value, i.e. $f_0 \in \mathbb{R}$
- $\hat{f}_n$ therefore gives us estimates of y

Classification: $f_0 = \mathbb{P}_0[Y = y|X_1, X_2, ..., X_p]$

- A vectored value, i.e. $f_0 = [p_1, p_2, ..., p_K] : p_j \in [0, 1], \sum_K p_j = 1$
- n.b. In a binary setting this simplies to a scalar, i.e. $f_0 = p_1 : p_1 = \mathbb{P}_0[Y = 1|X_1, X_2, ..., X_p] \in [0, 1]$
- $\hat{f}_n$ therefore gives us predictions of each class
- Can take the arg max, giving us Bayes Classifier

Bias variance decomposition

Let x_0 be a fixed point, $y_0 = f_0(x_0) + \epsilon$, and $\hat{f}_n$ be an estimate of f_0 from $(x_i, y_i) : i = 1, 2, ..., n$.

The MSE at x_0 can be decomposed as

$$\begin{aligned} MSE(x_0) &= \mathbb{E}_0[y_0 - \hat{f}_n(x_0)]^2 && (1) \\ &= \text{Var}(\hat{f}_n(x_0)) + \text{Bias}(\hat{f}_n(x_0))^2 + \text{Var}(\epsilon_0) && (2) \end{aligned}$$

Loss functions

Regression:

- MSE ($(y_i - \hat{y}_i)^2$)
- AIC, BIC, R^2, Adjusted R^2

Classification:

- Cross-entropy ($(y_i \log(\hat{p}_i)$)
- 0-1 loss ($\mathbb{I}(y_i \neq \hat{y}_i)$)
- Confusion matrix
- Receiver operating characteristic curve (& AUC)
- Gini index ($\sum_{m=1}^{|T|} q_m \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$)

Misc:

- Hinge loss ($\max(0, 1 - yf)$)

How do we estimate the test error?

- Our main technique is cross-validation.
- Different approaches:
  1. **Validation set:** Split the data in two parts, train the model on one subset, and compute the test error on the other.
  2. k-**fold:** Split the data into k subsets. Average the test errors computed using each subset as a validation set.
  3. **LOOCV:** k-fold cross validation with $k = n$.
- No approach is superior to all others.
- What are the main differences? How do the bias and variance of the test error estimates compare? Which methods depend on the random seed?

Evaluating a classification method

		Predicted class		
		− or Null	+ or Non-null	Total
True class	− or Null	True Neg. (TN)	False Pos. (FP)	N
	+ or Non-null	False Neg. (FN)	True Pos. (TP)	P
	Total	N*	P*	

We can calculate a number of statistics from this table, e.g.

- True positive rate (aka Sensitivity, aka Recall)

$$\mathbb{P}[\mathit{Predicted}\ + \mid \mathit{True}\, +], \text{i.e. } TP/P \tag{3}$$

Evaluating a classification method

		Predicted class		
		− or Null	+ or Non-null	Total
True class	− or Null	True Neg. (TN)	False Pos. (FP)	N
	+ or Non-null	False Neg. (FN)	True Pos. (TP)	P
	Total	N^*	P^*	

We can calculate a number of statistics from this table, e.g.

- True positive rate (aka Sensitivity, aka Recall)

$$\mathbb{P}[\textit{Predicted} + | \textit{True} +], \text{i.e. } TP/P \tag{3}$$

- True negative rate (aka Specificity)

$$\mathbb{P}[\textit{Predicted} - | \textit{True} -], \text{i.e. } TN/N \tag{4}$$

Evaluating a classification method

		Predicted class		
		− or Null	+ or Non-null	Total
True class	− or Null	True Neg. (TN)	False Pos. (FP)	N
	+ or Non-null	False Neg. (FN)	True Pos. (TP)	P
	Total	N*	P*	

We can calculate a number of statistics from this table, e.g.

- True positive rate (aka Sensitivity, aka Recall)

$$\mathbb{P}[\mathit{Predicted}\ + \,|\, \mathit{True}\, +], \text{i.e. } \ TP/P \tag{3}$$

- True negative rate (aka Specificity)

$$\mathbb{P}[\mathit{Predicted}\ - \,|\, \mathit{True}\, -], \text{i.e. } \ TN/N \tag{4}$$

- Positive predicted value (aka Precision)

$$\mathbb{P}[\mathit{True}\ + \,|\, \mathit{Predicted}\, +], \text{i.e. } \ TP/P^* \tag{5}$$

Evaluating a classification method

		Predicted class		
		− or Null	+ or Non-null	Total
True class	− or Null	True Neg. (TN)	False Pos. (FP)	N
	+ or Non-null	False Neg. (FN)	True Pos. (TP)	P
	Total	N^*	P^*	

We can calculate a number of statistics from this table, e.g.

- True positive rate (aka Sensitivity, aka Recall)

$$\mathbb{P}[\mathit{Predicted}\ + \,|\, \mathit{True}\, +], \text{i.e. } TP/P \tag{3}$$

- True negative rate (aka Specificity)

$$\mathbb{P}[\mathit{Predicted}\ - \,|\, \mathit{True}\, -], \text{i.e. } TN/N \tag{4}$$

- Positive predicted value (aka Precision)

$$\mathbb{P}[\mathit{True}\ + \,|\, \mathit{Predicted}\, +], \text{i.e. } TP/P^* \tag{5}$$

- Negative predicted value

$$\mathbb{P}[\mathit{True}\ - \,|\, \mathit{Predicted}\, -], \text{i.e. } TN/N^* \tag{6}$$

Classification losses

- The 0-1 loss or misclassification rate:

$$\sum_{m=1}^{|T|} \sum_{x_i \in R_m} \mathbf{1}(y_i \neq \hat{y}_{R_m})$$

Classification losses

- The 0-1 loss or misclassification rate:

$$\sum_{m=1}^{|T|} \sum_{x_i \in R_m} \mathbf{1}(y_i \neq \hat{y}_{R_m})$$

- The Gini index:

$$\sum_{m=1}^{|T|} q_m \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}),$$

where $\hat{p}_{m,k}$ is the proportion of class k within R_m, and q_m is the proportion of samples in R_m.

Classification losses

- The 0-1 loss or misclassification rate:

$$\sum_{m=1}^{|T|} \sum_{x_i \in R_m} \mathbf{1}(y_i \neq \hat{y}_{R_m})$$

- The Gini index:

$$\sum_{m=1}^{|T|} q_m \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}),$$

  where $\hat{p}_{m,k}$ is the proportion of class k within R_m, and q_m is the proportion of samples in R_m.

- The entropy:

$$-\sum_{m=1}^{|T|} q_m \sum_{k=1}^{K} \hat{p}_{mk} \log(\hat{p}_{mk}).$$

The Bootstrap

- **Main idea:** If we have enough data, the empirical distribution is similar to the actual distribution of the data.
- Resampling with replacement allows us to obtain pseudo-independent datasets.
- They can be used to:
  1. Approximate the standard error of a parameter (say, β in linear regression), which is just the standard deviation of the estimate when we repeat the procedure with many independent training sets.
  2. **Bagging**: By averaging the *predictions* $\hat{y}$ made with many independent data sets, we eliminate the variance of the predictor.

n.b. Can instead use the jackknife as a linear approximation.

Features / predictors / covariates

- (Non-linear) feature transformations
- Standardization
- Kernels
- Neural networks
- True, empirical, estimated distributions

Linear models

- Coefficients, standard errors, and hypothesis testing
- Interactions between predictors
- Non-linear relationships
- Correlation of error terms
- Non-constant variance of error (heteroskedasticity)
- Outliers
- High leverage points
- Collinearity
- Mis-specification

Regression methods

- Multiple linear regression
- Stepwise selection methods
- Ridge regression, Lasso, and elastic net
- Non-linear methods:
  - Polynomial regression
  - Cubic splines
  - Smoothing splines
  - Local regression
  - GAMs: Combining the above methods with multiple predictors
- Nearest neighbors regression
- Decision trees, Bagging, Random Forests, Boosting, and Neural Networks
- Neural Networks

Classification methods

- Nearest neighbors classification
- Naive Bayes
- Logistic regression
- LDA and QDA
- Stepwise selection methods
- Support vector classifier and support vector machines
- Decision trees, Bagging, Random Forests, Boosting
- Neural Networks

K-nearest neighbors

Mathematically, we can represent KNN as

K-nearest neighbors

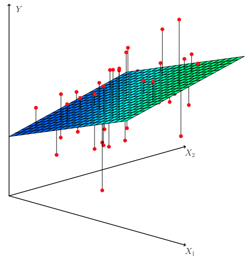
$$\mathbb{P}(Y = j | X = x_0) = \frac{1}{K} \sum_{i \in \mathcal{N}_0} \mathbb{I}(y_i = j) \tag{7}$$

We can apply Bayes rule to the resulting probabilities to get our classifier.

Multiple linear regression

Extension of linear regression to handle multiple predictors

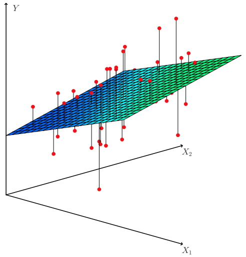
In multiple linear regression, we assume



$$\begin{aligned} Y &= \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \epsilon \\ \epsilon_i &\overset{iid}{\sim} \mathcal{N}(0, \sigma^2) \\ \mathbb{E}[Y|\mathbf{X}] &= \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots \end{aligned} \tag{8}$$

Multiple linear regression

Extension of linear regression to handle multiple predictors



In multiple linear regression, we assume

$$\begin{aligned} Y &= \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \epsilon \\ \epsilon_i &\overset{iid}{\sim} \mathcal{N}(0, \sigma^2) \\ \mathbb{E}[Y|\mathbf{X}] &= \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots \end{aligned} \tag{8}$$

In matrix notation:

$$\mathbb{E}[Y|\mathbf{X}] = \mathbf{X}\beta \tag{9}$$

where

$$\mathbf{X} = (1, X_1, X_2, ..., X_p) \tag{10}$$
$$\boldsymbol{\beta} = (\beta_0, \beta_1, ..., \beta_p)^\top \tag{11}$$

Logistic regression

An idea:
Let's apply a function to the result to keep it within $[0, 1]$

$$g^{-1}(z) = \frac{1}{1 + exp(-z)} \tag{12}$$

i.e.

$$\mathbb{P}[Y = 1|\mathbf{X}] = \frac{1}{1 + exp(-(\beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p))} \tag{13}$$

Logistic regression

An idea:
Let's apply a function to the result to keep it within $[0, 1]$

$$g^{-1}(z) = \frac{1}{1 + exp(-z)} \tag{12}$$

i.e.

$$\mathbb{P}[Y = 1|\mathbf{X}] = \frac{1}{1 + exp(-(\beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p))} \tag{13}$$

This is equivalent to modeling the log-odds, e.g.

$$\log\left[\frac{\mathbb{P}[Y = 1|\mathbf{X}]}{\mathbb{P}[Y = 0|\mathbf{X}]}\right] = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p \tag{14}$$

n.b. $exp(\beta_j)$ is commonly referred to as the *odds-ratio* for X_j

Ridge regression

Ridge regression solves the following optimization:

$$\min_{\beta} \quad \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{i,j} \right)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \tag{15}$$

In blue: the model RSS
In red: the squared ℓ_2 norm of β, or $\|\beta\|_2^2$

The parameter $\lambda > 0$ is a tuning parameter. It modulates the importance of fit vs. shrinkage.

- Typically determined via e.g. cross-validation

The Lasso

The **L**east **A**bsolute **S**hrinkage and **S**election **O**perator regression solves the following optimization:

$$\min_{\beta} \quad \sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{i,j}\right)^2 + \lambda \sum_{j=1}^{p}|\beta_j| \tag{16}$$

In blue: the model RSS

In red: the ℓ_1 norm of β, or $\|\beta\|_1$

The Lasso

The **L**east **A**bsolute **S**hrinkage and **S**election **O**perator regression solves the following optimization:

$$\min_{\beta} \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{i,j} \right)^2 + \lambda \sum_{j=1}^{p} |\beta_j| \qquad (16)$$

In blue: the model RSS
In red: the ℓ_1 norm of β, or $\|\beta\|_1$ **Note**: Unlike ridge regression, LASSO does not have a closed form solution.

Why would we use the Lasso instead of Ridge regression?

- Ridge regression shrinks all the coefficients to a non-zero value
- The Lasso shrinks some of the coefficients all the way to zero.
  - Similar to subset selection: will select variables for you

Cubic splines

Very popular, since they give very smooth predictions over X.

- Define a set of knots $\xi_1 < \xi_2 < \cdots < \xi_K$.
- We want the function $Y = f(X)$ to:
  1. Be a cubic polynomial between every pair of knots ξ_i, ξ_{i+1}.
  2. Be continuous at each knot.
  3. Have continuous first and second derivatives at each knot.

Fact: Given constraints, we need $K + 3$ basis functions:

$$f(X) = \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3 + \beta_4 h(X, \xi_1) + \cdots + \beta_{K+3} h(X, \xi_K) \tag{17}$$

where,

$$h(x, \xi) = \begin{cases} (x - \xi)^3 & \text{if } x > \xi \\ 0 & \text{otherwise} \end{cases}$$

Smoothing splines

Our goal is to find the function f which minimizes

$$\sum_{i=1}^{n}(y_i - f(x_i))^2 + \lambda \int f''(x)^2 dx$$

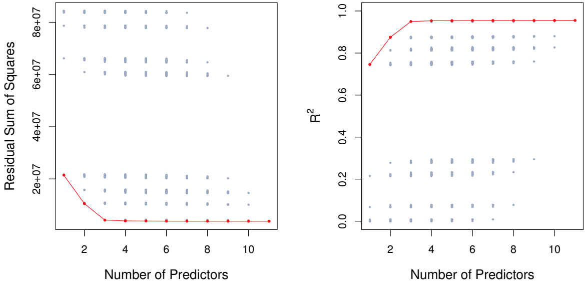
- The RSS of the model.
- A penalty for the roughness of the function.

For regularization, we have that $\lambda \in (0, \infty)$

- When $\lambda = 0$, f can be any function that interpolates the data.
- When $\lambda = \infty$, f will be the simple least squares fit

Kernel smoothing

Idea: Why not just use the subset of observations *closest* to the point we're predicting at?



- Observations averaged *locally* for predictions.
- Can use different weighting kernels, e.g.

$$K_\lambda(x_0, x) = D\left(\frac{|x - x_0|}{h_\lambda(x_0)}\right) \quad (18)$$

Generalized additive models

The extension of basis functions to multiple predictors (while maintaining additivity) , e.g.

Linear model

$$\texttt{wage} = \beta_0 + \beta_1\texttt{year} + \beta_2\texttt{age} + \beta_3\texttt{education} + \epsilon \tag{19}$$

Additive model

$$\texttt{wage} = \beta_0 + f_1(\texttt{year}) + f_2(\texttt{age}) + f_3(\texttt{education}) + \epsilon \tag{20}$$

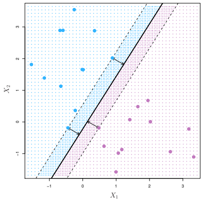
The functions $f_1, \ldots, f_p$ can be polynomials, natural splines, smoothing splines, local regressions, etc.

Support Vector Machines

$$\max_{\beta_0,\beta,\epsilon} M \tag{21}$$

subject to

- $\|\beta\| = 1$
- $y_i(\beta_0 + x_i^\top \beta) \geq M(1 - \epsilon_i) \; \forall \; i = 1, \ldots, n$
- $\epsilon_i \geq 0 \; \forall \; i = 1, \ldots, n$ and $\sum_{i=1}^n \epsilon_i \leq C$



Estimating the support vector classifier

Similar to the Maximal Margin Classifier:

- We can apply a Lagrange multipliers for our (constrained) optimization problem.
    - e.g. Karush-Kuhn-Tucker multipliers.
- This reduces our problem to finding $\alpha_1, \ldots, \alpha_n$ such that:

$$\max_{\alpha} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{i'=1}^{n} \alpha_i \alpha_{i'} y_i y_{i'} \underbrace{(x_i \cdot x_{i'})}_{\text{inner product}} \tag{22}$$

subject to

- $0 \leq \alpha_i \leq D \; \forall \; i = 1, \ldots, n$
- $\sum_i \alpha_i y_i = 0$

This only depends on the training sample inputs through the inner products $(x_i \cdot x_j)$ for every pair of points i, j

The kernel trick

Example: The polynomial kernel with $d = 2$ (and $p = 2$).

$$\begin{aligned} K(x,x') &= f(x,x') = \big(1 + \langle x, x' \rangle\big)^2 \\ &= (1 + x_1 x_1' + x_2 x_2')^2 \\ &= 1 + 2x_1 x_1' + 2x_2 x_2' + (x_1 x_1')^2 + (x_2 x_2')^2 + 2x_1 x_1' x_2 x_2' \\ &= 1 + \sqrt{2}x_1 \sqrt{2}x_1' + \sqrt{2}x_2 \sqrt{2}x_2' + x_1^2 (x_1')^2 + x_2^2 (x_2')^2 \\ &\quad + \sqrt{2}x_1 x_2 \sqrt{2}x_1' x_2' \end{aligned} \tag{23}$$

This is equivalent to the expansion:

$$\Phi(X) = (1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1 x_2)$$

giving us

$$K(i,k) = \langle \Phi(x_i), \Phi(x_k) \rangle \tag{24}$$

- Feature engineering is *"automated"* for us.
- Computing $K(x_i, x_k)$ directly is $O(p)$.

Decision Trees

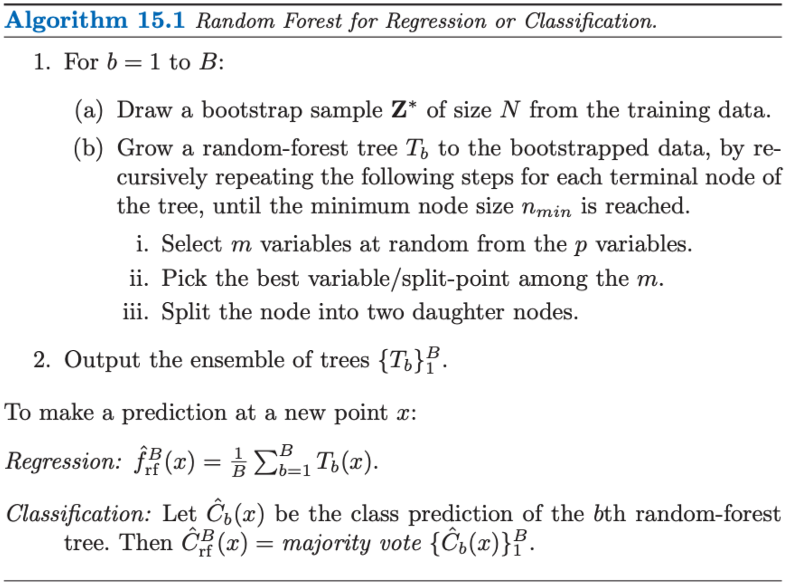
Using a *greedy* approach:

- Start with a single region R_1, and iterate:
  - Select a region R_k, a predictor X_j, and a splitting point s, such that splitting R_k with the criterion $X_j < s$ produces the largest decrease in RSS:

$$\sum_{m=1}^{|T|} \sum_{x_i \in R_m} (y_i - \bar{y}_{R_m})^2$$

  - Redefine the regions with this additional split.
- Terminate when there are 5 observations or fewer in each region.
- This grows the tree from the root towards the leaves.

Random Forest



Algorithm 15.1 *Random Forest for Regression or Classification.*

```
1. For $b = 1$ to $B$:
    (a) Draw a bootstrap sample $\mathbf{Z}^*$ of size $N$ from the training data.
    (b) Grow a random-forest tree $T_b$ to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size $n_{min}$ is reached.
        i. Select $m$ variables at random from the $p$ variables.
        ii. Pick the best variable/split-point among the $m$.
        iii. Split the node into two daughter nodes.
2. Output the ensemble of trees $\{T_b\}_1^B$.
```

To make a prediction at a new point x:

Regression: $\hat{f}_{\text{rf}}^B(x) = \frac{1}{B}\sum_{b=1}^B T_b(x)$.

Classification: Let $\hat{C}_b(x)$ be the class prediction of the bth random-forest tree. Then $\hat{C}_{\text{rf}}^B(x) = \textit{majority vote } \{\hat{C}_b(x)\}_1^B$.

Boosting

Boosting uses a set of weak learners (e.g. decision trees) to create a strong one.

The general algorithm is:

1. Fit an initial $\hat{f}_n^0$ to the data and compute residuals r_i.
2. For $b = 1, ..., B$:
   - Fit a weak leaner $\hat{f}_n^b$ on the residuals.
   - With learning rate λ_b, update prediction to:
   $$\hat{f}_n \leftarrow \hat{f}_n + \lambda_b \hat{f}_n^b. \tag{25}$$
   - Update the residuals
   $$r_i \leftarrow r_i - \lambda_b \hat{f}_n^b(x_i). \tag{26}$$
3. Output prediction, e.g.
$$\hat{f}_n(x) = \hat{f}_n^0 + \sum_{b=1}^{B} \lambda_b \hat{f}_n^b(x). \tag{27}$$

Neural networks

Neural networks are simply a generalization of the logistic regression case, e.g. for

$$\mathbb{P}(Y = 1|\mathbf{X}) = \sigma(\sigma(\mathbf{X}\boldsymbol{W}_1)\boldsymbol{W}_2) \tag{28}$$

Neural networks

Neural networks are simply a generalization of the logistic regression case, e.g. for

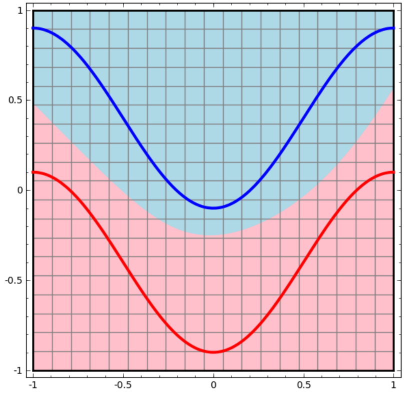
$$\mathbb{P}(Y = 1|\mathbf{X}) = \sigma(\sigma(\mathbf{X}\boldsymbol{W}_1)\boldsymbol{W}_2) \tag{28}$$

Our loss is

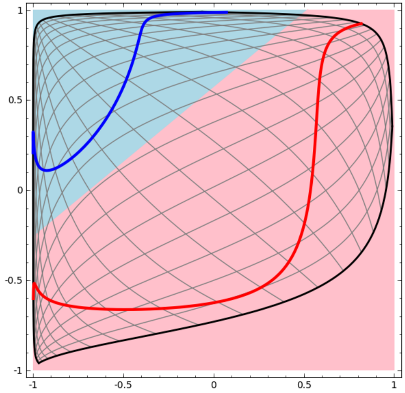
$$\begin{aligned} L(y_i, f(\mathbf{X}_i)) &= -y_i \log(p_i) - (1 - y_i)\log(1 - p_i), \text{ where} && (29) \\ p_i &= \frac{1}{1 + exp(-Z_{2,i})} && (30) \\ Z_{2,i} &= h_i \boldsymbol{W}_2 && (31) \\ h_i &= \frac{1}{1 + exp(-Z_{1,i})} && (32) \\ Z_{1,i} &= \mathbf{X}\boldsymbol{W}_1 && (33) \end{aligned}$$

Feature representation

How do the feature transformations get learned?



Original representation of curves



Hidden layer representation of curves

Well demonstrated by *Chris Olah's blog*.

The Transformer - model architecture

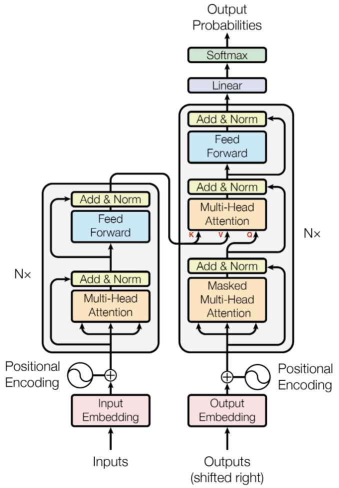
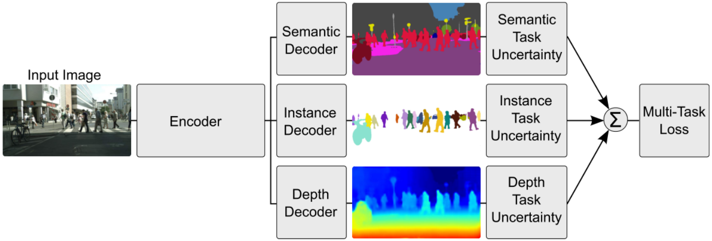


Figure 1: The Transformer - model architecture.

n.b. Visualization available *here*.

Model generalizations

Neural networks can be applied over multiple tasks (i.e. multi-task learning), e.g.



Kendall et al. 2017's multi-task model

Survival analysis

Analyzing right censored survival time

- Our observed time is $Y = \min(T, C)$
- We have an associated indicator $\delta = \mathbb{I}(T \leq C)$

Two commonly used estimators

- Kaplan Meier Estimator: estimates the survival function for a small number of groups
  - Can use log-rank test to confirm statistical significance.
- Cox-proportional hazards: assumes proportionality in the hazard functions.
  - Similar to (pooled) logistic regression (breaking follow-up time into individual time ranges)

Self testing questions

For each of the regression and classification methods:

1. What are we trying to optimize?
2. What does the fitting algorithm consist of, roughly?
3. What are the tuning parameters, if any?
4. How is the method related to other methods, mathematically and in terms of bias, variance?
5. How does rescaling or transforming the variables affect the method?
6. In what situations does this method work well? What are its limitations?