A TOUR OF MACHINE LEARNING CLASSIFIERS USING SCIKIT-LEARN

- NO single classifier works best across all possible scenarios
- in practice, it is recommended that you compare the performance of at least a handful of different algorithms & select the best model for the particular problem.

important factors:

- # of features
- amount of noise
- whether classes are linearly separable.

- 5 main steps for training:
  1. feature selection & collection of labeled training data.
  2. choosing a performance metric (MSE vs R^2)
  3. choosing an algorithm & training the model
  4. Evaluate performance of the model
  5. tune model performance.

- train/test split from sklearn pre-shuffles the dataset.
- Stratification: train/test split will return splits that have the same proportions of class labels as the input dataset

*also called z-score normalization

- Standard scaler: standard scaler uses the fit method to estimate the mean & standard deviation of each feature, where transform actually makes features have a mean of zero & a standard dev of 1.

- one-vs-rest: the model handles multi-classification tasks by creating n binary classifiers where n is the # of unique classes, that way for the given class, the the desired class is given positive outcomes, rest are given negative.

- Error vs. Accuracy

  Error = # of incorrect predictions / # of predictions

  Accuracy: (1 - Error) x 100 (%)

- score method: using the score method combines the predict / accuracy score call

- perceptron will never converge if the classes are not linearly seperable

LOGISTIC REGRESSION

- LR works based on defining odds & the probability of a "positive event"

ODDS = $p/(1-p)$, where p = probability of a positive event

label

features

$$p := p(y=1 \mid x)$$

natural log

$$\text{logit}(p) = \log\left(\frac{p}{1-p}\right)$$

- under a logistic model we assume a linear relationship between our inputs & log-odds:

$$\text{logit}(p) = w_1x_1 + w_2x_2 + \cdots + w_mx_m + b = \sum_{i=j} w_jx_j + b = w^Tx + b$$

we are interested in solving for p.
so we take in inverse nat. log to find $\sigma(z)$.

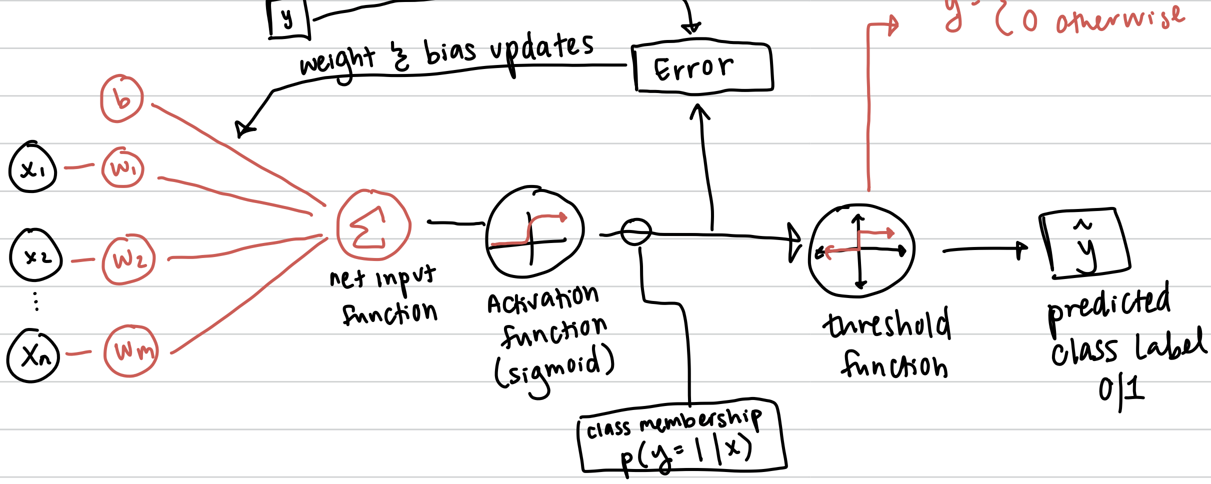
$$\sigma(z) = \frac{1}{1+e^{-z}}, \quad \text{where } z = w^Tx + b$$

this is called the logistic sigmoid function.

if $z \to \infty$, $\sigma(z) = 1$ $(e^{-\infty} \to 0)$. similarly, if $z \to -\infty$, $\sigma(z) = 0$

- this works similar Adaline, where instead of an identity function, we use a sigmoid activation function

$$\hat{y} = \begin{cases} 1 & \text{if } \sigma(z) \geq 0.5 \\ 0 & \text{otherwise} \end{cases}$$



LEARNING MODEL WEIGHTS VIA logistic loss FUNCTION

ADALine MSE loss:

$$L(w,b|x) = \sum_i \frac{1}{2}\left(\sigma(z^i) - y^i\right)^2$$

$p(Y=1 \mid x = x^i) = \sigma(z^i)$

$p(Y=0 \mid x = x^i) = 1 - \sigma(z^i)$

- Assuming our examples are independent, we can define the likelihood $\boxed{\mathcal{L}}$ that we want to maximize in logistic regression:

Prob. mass func.

$$\mathcal{L}(w,b|x) = p(y|x; w,b) = \overset{\text{product}}{\prod_{i=1}^{n}} p(y^i|x^i; w,b) = \prod_{i=1}^{n} \underbrace{\left(\sigma(z^i)\right)^{\boxed{y^i}\,\text{class}} \cdot \left(1-\sigma(z^i)\right)^{\boxed{1-y^i}\,\text{not class}}}_{\text{Bernoulli principal}}$$

- its much easier to maximize the nat. log of this function, which is called the log-likelihood:

$$l(w,b|x) = \log \mathcal{L}(w,b|x) = \sum_{i=1} \left[y^i \log(\sigma(z^i)) + (1-y^i) \cdot \log(\sigma(z^i)) \right]$$

✲ we can maximize this using gradient ascent ✲

maximize = Add

Minimize = Subtract

For gradient descent:

$$\mathcal{L}(w,b) = \sum_{i=1}^{n} \left[-y^i \log(\sigma(z^i)) - (1-y^i) \log(1-\sigma(z^i)) \right]$$

$$\therefore \ \mathcal{L}(\sigma(z), y; w,b) = -y \log(\sigma(z)) - (1-y) \log(1-\sigma(z))$$

$$\mathcal{L}(\sigma(z), y; w,b) = \begin{cases} -\log(\sigma(z)) & \text{if } y=1 \\ -\log(1-\sigma(z)) & \text{if } y=0 \end{cases}$$

TACKLING OVERFITTING VIA REGULARIZATION

- overfitting means that the model is too well fit for the training data, & doesnt generalize well to unseen data.

- If the model is overfit, the model will have high variance, which is caused by having too many parameters (too complex for the data)

- if the model is underfit, it will have high bias, meaning the data is too complex for the model

* this is called the bias/variance tradeoff *

- to find a bias-variance tradeoff, we can tune the complexity of the model via Regularization.

- Regularization is very useful method for handling collinearity (large correlation between features), filtering out noise, & preventing overfitting.

eigenvector

L2 regularization: $\frac{\lambda}{2 \cdot n} ||w||^2 = \frac{\lambda}{2n} \sum_{j=1}^{m} w_j^2$

HIGHLY SENSITIVE TO SCALING

loss changes to:

$$\mathcal{L}(w,b) = \frac{1}{n} \sum_{i=1}^{n} [-y^i \log(\sigma(z^i)) - (1-y^i)\log(1-\sigma(z^i)) + \frac{\lambda}{2n} ||w||^2$$

- Regularization is useful to prevent overfitting, but if the strength is too great, it will also lead to underfitting.