Machine Learning

Earl Wong

ML Models and Theory

- Linear Model: $y = w_1 x_1 + w_2 x_2 + \ldots + w_n x_n + b, \quad (x_1, x_2, \ldots x_n, y)$
- This model forms the foundation of machine learning.
- This model describes an n dimensional hyperplane.
- A hyperplane is a flat linear manifold (=surface).

ML Models and Theory

- Intuition: n = 1, the model describes a line (= 1 dimensional hyperplane)

- Intuition: n = 2, the model describes a plane (= 2 dimensional hyperplane)

- Intuition: n = m, the model describes an m dimensional hyperplane.

ML Models and Theory

- Goal: Find the weights {w} that scale the features {x} to minimize a residual error.
- i.e. Find the best fit hyperplane.

Three common loss functions are employed:

- 1) vanilla regression (least squares) , 2) ridge regression (least squares + L2 regularization) , 3) lasso (least squares + L1 regularization)

ML Models and Theory

$$L_{\textbf{regression}}(w) = \frac{1}{2n} \sum_{i=1}^{n} (y_i - w^T x_i)^2$$

$$L_{\textbf{ridge}}(w) = \frac{1}{2n} \sum_{i=1}^{n} (y_i - w^T x_i)^2 \; + \; \lambda \sum_{j=1}^{d} w_j^2$$

$$L_{\textbf{lasso}}(w) = \frac{1}{2n} \sum_{i=1}^{n} (y_i - w^T x_i)^2 \; + \; \lambda \sum_{j=1}^{d} |w_j|$$

ML Models and Theory

- Ridge regression and lasso add regularization terms to the standard vanilla regression loss term.

- Ridge regression encourages smaller weights, due the L2 penalty term.

- Lasso encourages sparse weights, due to the the L1 penalty term.

- If vanilla regression results in excessively large weights for certain features, ridge regression might be the better alternative.

- If we suspect that only a subset of the features play an important role, lasso might be the better alternative, since it encourages sparsity = weights that are 0.

ML Model and Theory

- Each loss function results in a bias variance tradeoff.
- Suppose a large dataset is divided into 5 smaller, disjoint datasets.
- When training the 5 model using vanilla regression, large variations in the 5 models can result due to outlier data.
- i.e. The weights are allowed to vary unconstrained.
- However, there is little bias in the resulting model, because there are no constraints on the weights.

ML Model and Theory

- In contrast, a model trained using ridge regression places a L2 regularization on the weights.

- The 5 resulting models now have smaller variations with respect to each other.

- i.e. Outlier data is “neutralized”.

- However, a bias in the output models has resulted, since the weights can no longer freely vary and “float” to arbitrarily large values.

ML Model and Theory

- A model trained using lasso places a L1 regularization on the weights.

- The 5 resulting models now have smaller variations with respect to each other.

- Once again, a bias in the output models has resulted, since many of the resulting weights are not just small, but have the value of 0.

ML Model and Theory

Bias - variance tradeoff

- Vanilla regression - high variance, but little bias
- Ridge regression - lower variance, but more bias
- Lasso - lower variance, but more bias
- Depending on the nature of the features, ridge and lasso will outperform each other.
- If the true model requires all of the features to be present, ridge will outperform lasso and vice versa.

A Transition to Binary Classifiers

- The current regression output - the hyperplane - can be easily converted to a classifier.

- For n = 1, any output lying above the line can be associated with one class, while any output lying below the line can be associated with a different class.

- For n = 2, any output lying above the plane can be associated with one class, while any output lying below the plane can be associated with a different class.

- The same is true for n = m, but cannot be easily visualized for hyperplanes with dim(n) > 2.

A Transition to Binary Classifiers

- Given: $y = w^T x + b$

- We now introduce the following additional equation, transforming our regression result into a classifier:

$$\hat{y} = \begin{cases} 1 & \textbf{if } f(x) > 0, \\ 0 & \textbf{otherwise} \end{cases}$$

The Perceptron

- The perceptron was an algorithm that employed an update rule to adjust the weights and bias of a linear model, to perform classification.
- The class labels are now given as {-1, 1}.
- The classifier output is now computed as: $\hat{y} = \textbf{sign}(w^T x + b)$
- The update rule is: $w \leftarrow w + \eta\, y_i\, x$ $b \leftarrow b + \eta\, y_i$ η *denotes the learning rate .*
- Note: The update rule is ONLY applied to incorrectly classified output.

The Perceptron

The main assumption of the Perceptron

- The algorithm assumes that the data can be divided into 2 different classes (=linearly separable) via some hyperplane.
- If the assumption is violated, the algorithm will never converge, since no such hyperplane exists.
- In practice, various heuristics can be applied, to produce an acceptable result even when the data is not fully separable.

Support Vector Machine (SVM)

- SVM's extend the objective of the Perceptron, by finding not just a separating hyperplane, but the best separating hyperplane.

- Best is defined as the hyperplane with the largest margin (= separation distance) between the nearest point of each class.

- The distance from an n dimensional data point to the hyperplane is given by:

$$\textbf{dist}(x_i) = \frac{|w^T x_i + b|}{\|w\|}$$

- Leading to the optimization formulation: $\max_{w,b} \min_{i} \frac{|w^T x_i + b|}{\|w\|}$

Support Vector Machine (SVM)

- The underlying constraint in the optimization problem is:

minimize $\frac{1}{2}\|w\|^2$ **subject to** $y_i(w^T x_i + b) \geq 1, \forall i$

- The margin is maximized by minimizing the norm of the weights.

- This occurs, because the norm of the weights is in the denominator of the optimization equation.

SVM

- In practice, the data may ALSO not be absolutely separable.
- Should this be true, slack variables are introduced, to allow misclassification errors.

Logistic Regression

- With logistic regression, a probabilistic framework is taken.
- The output of the linear model is now passed through a sigmoid function.
- The formulation is: $p(y = 1 \mid x) = \sigma(w^\top x + b)$
- For sigma: $\sigma(y) = \dfrac{1}{1 + e^{-y}}$
- If the resulting probability is greater than 0.5, class 1 is predicted - otherwise, class 0 is predicted.

Logistic Regression

- The linear model weights are updated, using the following loss function:

$$L_{\textbf{logistic}}(w) = -\sum_{i=1}^{m}\left[y_i \log \sigma(w^\top x_i + b) + (1 - y_i)\log\left(1 - \sigma(w^\top x_i + b)\right)\right]$$

- Here, x_i denotes one sample of our input feature vector.

- As shown, we are summing the result from m different input feature vector samples.

Logistic Regression

- It is useful, to understand what is back propagated into the linear model.

- Hence, we will now show the derivation.

$$\frac{\partial L}{\partial z} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \qquad \hat{y} = \sigma(z) = \frac{1}{1 + e^{-z}}$$

$$L_{\textbf{logistic}}(w) = \left[y \log \hat{y} + (1 - y) \log\left(1 - \hat{y}\right) \right]$$

$$L_{\textbf{logistic}}(w) = \left[y \log \sigma(z) + (1 - y) \log\left(1 - \sigma(z)\right) \right]$$

$$L_{\textbf{logistic}}(w) = \left[y \log \sigma(w^\top x_i + b) + (1 - y) \log\left(1 - \sigma(w^\top x_i + b)\right) \right]$$

Logistic Regression

$$\frac{\partial \hat{y}}{\partial z} = \frac{\partial\left(\frac{1}{1+e^{-z}}\right)}{\partial z} = \frac{-e^{-z}}{(1+e^{-z})^2} = \sigma(z)(1-\sigma(z))$$

$$\frac{\partial L}{\partial \hat{y}} = \frac{\partial\left[-(y\log\hat{y} + (1-y)\log\left(1-\hat{y}\right)\right)\right]}{\partial \hat{y}} = \frac{-y}{\hat{y}} + (1-y)\frac{1}{1-\hat{y}}$$

$$\frac{\partial L}{\partial \hat{y}} = \frac{\partial\left[-(y\log\sigma(z) + (1-y)\log\left(1-\sigma(z)\right)\right)\right]}{\partial \sigma(z)} = \frac{-y}{\sigma(z)} + (1-y)\frac{1}{1-\sigma(z)}$$

$$\frac{\partial L}{\partial z} = \frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial z} = \left(\frac{-y}{\sigma(z)} + (1-y)\frac{1}{1-\sigma(z)}\right)\left(\sigma(z)(1-\sigma(z))\right) = -y(1-\sigma(z)) + (1-y)\sigma(z))$$

$$\frac{\partial L}{\partial z} = -y + \sigma(z) = \hat{y} - y, \textit{ where } y \textit{ is the true classifier value, } y \in \{0,1\}$$

Extension

- We showed the derivation for logistic regression (aka binary classifier).
- The more general cross entropy loss, used for a n output classifier, follows a similar process.
- Now, instead of a sigmoid function, a softmax is employed, to convert the output logits to probabilities.
- Now, an output vector containing each class location, is back propagated, to update the network weights.

Extension

- Output vector = [0.2, 0.3, -0.6, 0.05, 0.05], for class 3.
- Positive numbers decrease the future predicted values for classes 1, 2, 4 and 5.
- The negative number increases the future predicted value for class 3.
- If the probability for class 3 is already high, the negative number will be small, resulting in a small change in weight values.
- Conversely, if the probability for class 3 is small, the negative number will be large, resulting in a large change in weight values.

Summary

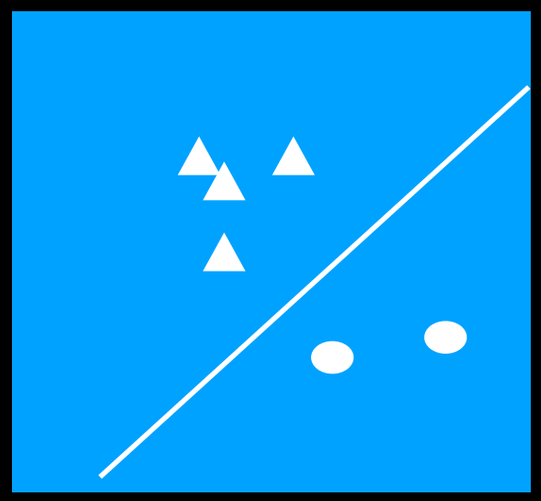
- We are given a linear model.
- From the model, we can train binary classifiers by employing different training methods: regression+sigmoid = logistic regression, perceptron, SVM.
- In practice, logistic regression is the most powerful, since it trains with probabilistic outputs.
- Probabilistic outputs enable training via confidence weighting.
- In contrast, perceptron and SVM operate on raw outputs = without confidence weighting.

Summary

- Up to now, we have been working with a single hyperplane in n dimensional space.
- For regression, we fit the best hyperplane to minimize residual error.
- For classification, we determined the best hyperplane to separate two distinct classes.
- But, what if our data cannot be subdivided naturally, with a hyperplane?

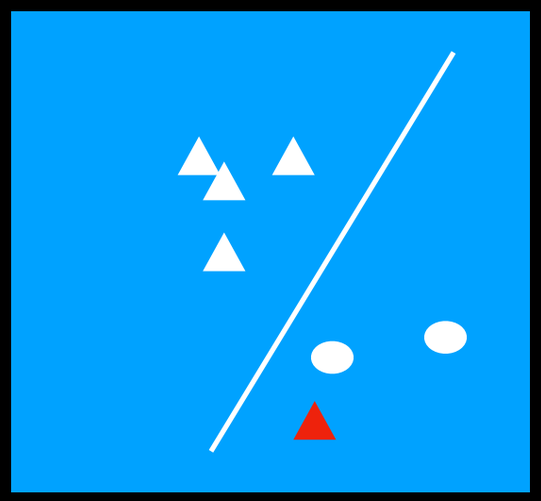
Summary

Case1



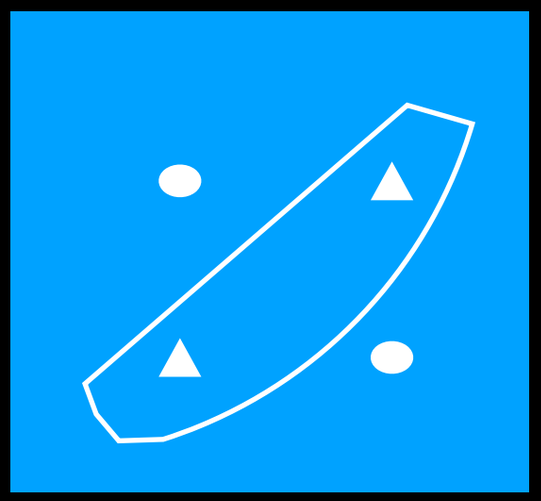
Ideal, separable case (vanilla SVM, vanilla perceptron, logistic regression)

Case2



Accept some errors (SVM with slack variables, perceptron with heuristics, logistic regression with errors)

Case3



Move beyond a single hyperplane Or transform to a non-linear space

Classification - Beyond Linear Models

- Decision Trees
- SVM and Kernels
- Linear models with ReLU

Decision Trees

- Decision trees can perform classification (and regression), without the need for a mathematical model.
- At every parent node in a tree, two child nodes are created, based on a pre-defined criteria.
- The criteria is based on impurity and information gain.
- Impurity is measured using the gini index or entropy.
- Information gain measures the decrease in impurity, resulting from the chosen split.

Decision Tree

- Every conceivable split is examined.
- The split that maximizes information gain (=maximizes the decrease in impurity) is selected.
- This leads to a decision rule for every node, based on one of the input features.
- The process (then) repeats for subsequent child nodes, etc.

Decision Trees

- There are 2 flavors of decision trees: Bagging and Boosting

- Because the splits are data driven (=highly sensitive to the training data), there can be huge variations (=high variance) in the resulting output.

- Bagging creates an ensemble of trees, and averages the results from the ensemble - thereby reducing the variance.

- For classification, majority vote is used.

Decision Trees

- In contrast, Boosting creates a shallow tree / a stub = weak learner.

- The samples that fail to be classified correctly, are then fed into another shallow tree = weak learner.

- The process repeats, until there are no classification errors left, or, after a finite number of weak learners have been created / cascaded.

- Boosting can be viewed as a process that reduces the residual error with each cascade, by sending the failure cases through a new weak learner.

Decision Tree

- By construction, decision trees are highly non-linear constructs.

- Decision trees subdivide the input feature space using decision rules based on hard thresholds applied to a specific input feature.

SVM

- Support vector machines can also perform accurate classification on data that cannot be divided by a simple hyperplane.

- SVM's do this by transforming the input feature data from a linear space into a non-linear space.

- In the non-linear space, a simple hyperplane will now suffice.

- In practice, SVM's employ the "kernel" trick to avoid the heavy mathematical computation, need to transform the data into it's new non-linear space.

Logistic Regression and ReLU

- Suppose we begin with a linear model:

$$y_1 = W_1 x + b_1$$

- And stack another linear layer:

$$y_2 = W_2(y_1) + b_2 = W_2(W_1 x + b_1) + b_2 = Wx + b$$

- The result is still linear.

- i.e. The expressive power of the 2 stacked layers has not increased.

Logistic Regression and ReLU

- In order to increase expressive power, non-linear activations need to be introduced in-between the linear layers.

$$h_1 = ReLU(W_1 x + b_1)$$
$$h_2 = ReLU(W_2 h_1 + b_2) \qquad ReLU(x) = max(0,x)$$
$$out = W_3 h_2 + b_3$$

- ReLU acts as a hinge, allowing the input space to fold.
- ReLU chops the input space into many smaller linear cells.
- As the number of layers increase, complex boundaries result.
- ReLU's are nice to work with, since the derivative of ReLU(x) is a constant, making it highly amenable to back propagation.

Function Approximation

- Neural networks are said to be universal function approximators.
- In practice, the networks can grow in depth or width, to achieve this objective.
- Amongst the two directions, depth provides several significant advantages - faster expressivity being one such advantage.
- This has been demonstrated through empirical testing.

FC/Linear Model to CNN

- With FC layers, everything is connected to everything.
- For example, if the input is a 16x16 (= 256) pixel image, the image is first unrolled into a 256 length vector.
- Then, 256 weight vectors are employed, to execute the linear model.
- When the FC layers are stacked, even more weights are used, since everything is connected to everything.

FC/Linear Model to CNN

- With a CNN layer, a single convolutional filter/weight vector is learned.
- This weight vector is then applied locally to all of the pixels in the image.
- Because of weight sharing, the number of needed weights decreases tremendously, relative to its FC counterpart.
- Typically, the weight vector is a fixed nxn (typically n = 3) convolutional filter.

FC/Linear Model vs CNN

What is lost?

- Unlike the FC layer, the receptive field is no longer the entire input image.
- Instead, it is a local 3x3 patch.
- i.e. The CNN filter creates a local linear layer on a small input patch, by applying a fixed filter to all spatial locations.

FC/Linear to CNN

What is gained?

- The number of needed weights has been greatly reduced.
- An output feature map associated with the convolutional filter is produced.
- When the CNN layers are stacked, hierarchical features will be produced.
- The learned CNN filters in the early layers will produce maps with coarse features.
- The learned CNN filters in the final layers will produce maps with semantically meaningful features.

Function Approximation

- However, training deep networks introduces it's own set of problems.
- Specifically, vanishing or exploding gradients become a problem.
- To mitigate these issues, residual connections/skip connections are used.
- To mitigate these issues, batch norm is employed.
- To mitigate these issues, ReLU activation functions are used.

Function Approximation

- In addition, network initialization becomes important.
- In addition, the learning rate recipe becomes important.
- When all of these tricks are employed successfully, a deep network is learned.
- This network can achieve powerful expressivity, and produce valuable hierarchical representations.

FC vs CNN

- FC->Batch Norm->ReLU—>FC->Batch Norm->ReLU
- CNN-> Batch Norm->ReLU—>CNN->Batch Norm->ReLU

What Follows

- Transformers
- Mamba

What Follows

- Transformers add the feature of attention.
- i.e. Everything can “attended” to something else, regardless of spatial location/spatial position.
- In contrast, CNN’s cannot accomplish this, if the effective receptive field is not large enough.
- For example: If the image is 16x16 and you stack two 3x3 convolutional filters, the effective receptive field is only 5x5.

What Follows

What is the downside of transformers?

- Because every input token can attend to every other input token, the computational complexity is quadratic.

- Because every input token can attend to every other input token, the memory requirements are huge.

- However, as of present day 2025, transformers are the backbone of current foundational models.

What Follows

- Mamba is rooted in the theory of dynamical systems.
- Mamba employs state space layers.
- Instead of remembering everything (like transformers), mamba continually updates what it learns, retaining the most important information.
- By construction, Mamba runs in linear time.
- In the research environment, Mamba has been shown to outperform transformers, for very long input token sequences.

The Future

- Edge devices - CNN’s (and LSTM)
- Foundational Models - Currently Transformers
- Foundational Models with long input token sequences - Mamba

The Journey

- Linear models and Decision Trees - minimal/limited data, good results
- CNN - a lot more data, more powerful results
- Transformers - BIG, BIG data. Everything works better, the bigger the transformer becomes.
- i.e. Small transformers can’t outperform CNN’s. But, large transformers can and do.
- Mamba - will win the long token sequence game, since it’s big O is linear, and not quadratic.

AGI

- Not there.
- LLM's have "upped" the game though, because language provides an important unifying element/entity.
- For AGI, agents need to interact in a World Model.
- Then, a new set/level of interesting capabilities will emerge, when the (needed) "intelligent optimization algorithm" is learned.