Regularizations

L1 Regularization – Lasso Regression

$$\sum_{i=1}^{n}\left(y_i - \sum_{j=1}^{p} x_{ij}\beta_j\right)^2 + \lambda\sum_{j=1}^{p}|\beta_j|$$

- Works well for *feature selection* in case we have a huge number of features. L1 penalty tends to pick one variable at random when predictor variables are correlated.

L2 Regularization – Ridge Regression

$$\sum_{i=1}^{n}\left(y_i - \sum_{j=1}^{p} x_{ij}\beta_j\right)^2 + \lambda\sum_{j=1}^{p}\beta_j^2$$

- If λ is very large then it will add too much weight and it will lead to under-fitting
- This technique works very well *to avoid over-fitting issue*

Note: the regularization methods can also be applied on logistic regression models. L1 versus L2 – See Ace the Data Science Interview Page 130 Solution #7.20.

Logistic Regressions – sigmoid & softmax functions

In the two-class logistic regression, the predicted probabilities are as follows, using **the sigmoid function**:

$$P(Y_i = 1) = \frac{1}{1 + e^{-\beta\cdot X_i}}$$

In the multiclass logistic regression, with K classes, the predicted probabilities are as follows, using **the softmax function**:

$$P(Y_i = k) = \frac{e^{\beta_k\cdot X_i}}{\sum_{0\le c\le K} e^{\beta_c\cdot X_i}}$$

Variance-Bias Trade-off

http://scott.fortmann-roe.com/docs/BiasVariance.html

Conceptual Definitions

- **Error due to Bias**: The error due to bias is taken as *the difference between the expected (or average) prediction of our model and the correct value* which we are trying to predict. Of course you only have one model so talking about expected or average prediction values might seem a little strange. However, imagine you could repeat the whole model building process more than once: each time you gather new data and run a new analysis creating a new model. Due to randomness in the underlying data sets, the resulting models will have a range of predictions. Bias measures how far off in general these models' predictions are from the correct value.

- **Error due to Variance**: The error due to variance is taken as *the variability of a model prediction for a given data point*. Again, imagine you can repeat the entire model building process multiple times. The variance is how much the predictions for a given point vary between different realizations of the model.

Mathematical Definition

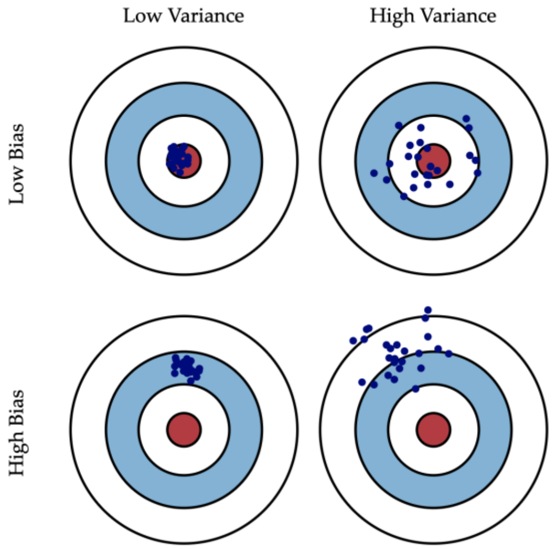
We denote the variable we want to predict as Y and the covariates as X. we may assume that there is a relationship relating one to the other such as $Y = f(X) + \epsilon$ and the error term is normally distributed as $\epsilon \sim N(0, \sigma_e^2)$.

We might estimate a model $\hat{f}(x)$ of $f(x)$ using linear regression or another modelling technique.

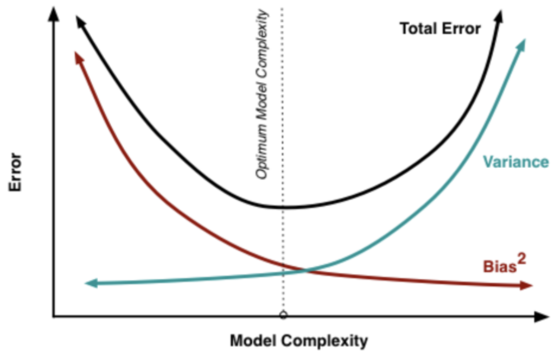
$$Err(x) = E\left[\left(Y - \hat{f}(x)\right)^2\right] = \left(E[\hat{f}(x)] - f(x)\right)^2 + E\left[\left(\hat{f}(x) - E[\hat{f}(x)]\right)^2\right] + \sigma_e^2$$

$$Err(x) = Bias^2 + Variance + Irreducible\ Error$$

Graphical Illustration



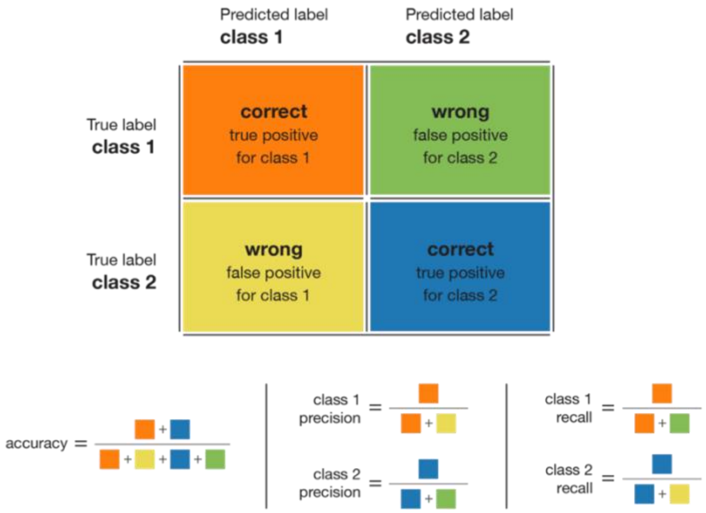
At its root, dealing with bias and variance is really about dealing with over- and under-fitting. *Bias is reduced and variance is increased in relation to model complexity*. As more and more parameters are added to a model, the complexity of the model rises and variance becomes our primary concern while bias steadily falls. For example, as more polynomial terms are added to a linear regression, the greater the resulting model's complexity will be.



Some More Examples

- Squiggly line (overfitted)
  - Low bias
  - High variance – it results in vastly different sums of squares for different datasets
- Straight line (underfitted)
  - High bias – can't capture the relations
  - Low variance – it makes consistently good predictions

Confusion Matrix & ROC & AUC



	Predicted label class 1	Predicted label class 2
True label class 1	correct true positive for class 1	wrong false positive for class 2
True label class 2	wrong false positive for class 1	correct true positive for class 2

accuracy = (🟧 + 🟦) / (🟧 + 🟨 + 🟦 + 🟩)
class 1 precision = 🟧 / (🟧 + 🟨)
class 2 precision = 🟦 / (🟦 + 🟩)
class 1 recall = 🟧 / (🟧 + 🟩)
class 2 recall = 🟦 / (🟦 + 🟨)

- The **precision** of a class define *how trustable* is the result when the model answer that a point belongs to that class. **Pr**ecision: True Positive/**Pr**edicted Positive
- The **recall** of a class expresses *how well* the model is able to detect that class/what proportion of samples are correctly classified. Low recall value means our model is not doing well for this class. **Re**call: TP/**Re**al Positive
- The **accuracy** of the model is basically the total number of correct predictions divided by total number of predictions.

- The **F1** score of a class is given by the harmonic mean of precision and recall (2×precision×recall / (precision + recall)), it combines precision and recall of a class in one metric.
- *Precision is related to the type I error rate, but in a slightly more complicated way, as it also depends upon the prior distribution of seeing a relevant vs an irrelevant item. (Link)*

For a given class, the different combinations of recall and precision have the following meanings :

- high recall + high precision : the class is perfectly handled by the model
- low recall + high precision : the model can't detect the class well but is highly trustable when it does
- high recall + low precision : the class is well detected but the model also include points of other classes in it
- low recall + low precision : the class is poorly handled by the model

F1-Score vs. Accuracy

Accuracy:

Pro: *Easy to interpret. If we say that a model is 90% accurate, we know that it correctly classified 90% of observations.*
Con: *Does not take into account how the data is distributed.* For example, suppose 90% of all players do not get drafted into the NBA. If we have a model that simply predicts every player to not get drafted, the model would correctly predict the outcome for 90% of the players. This value seems high, but the model is actually unable to correctly predict any player who gets drafted.

F1-score:

Pro: Takes into account how the data is distributed. For example, if the data is highly imbalanced (e.g. 90% of all players do not get drafted and 10% do get drafted) then F1-score will provide a better assessment of model performance.
Con: Harder to interpret. The F1-score is a blend of the precision and recall of the model, which makes it a bit harder to interpret.

As a rule of thumb:

- We often use accuracy when *the classes are balanced* and there is no major downside to predicting false negatives.
- We often use F1 score when *the classes are imbalanced* and there is a *serious downside to predicting false negatives* *(? This item might not be true as recall is more important).*
- When working on problems *with heavily imbalanced datasets* AND *you care more about detecting positives than detecting negatives* (outlier detection / anomaly detection) then you would prefer the F1-score more.

For example, if we use a logistic regression model to predict whether or not someone has cancer, *false negatives* are really bad (e.g. predicting that someone does not have cancer when they actually do) so F1 score (?) will penalize models that have too many false negatives more than accuracy will.

Note: when there is a serious downside to predicting false negative, such as to detect a cancer or a transaction fraud, use *Recall* as the metric (VISA Sr DS interview as of 22/7).

ROC & AUC

Each value of the threshold T generates a point (false positive, true positive) and, then, the ROC curve is the curve described by the ensemble of points generated when the threshold T varies from 1 to 0.

- y-axis True Positive Rate = Recall
- x-axis False Positive Rate = 1 – Precision

ROC curves make it easy to identify the best threshold for making a decision; while the **AUC** helps you decide which categorization method is better.

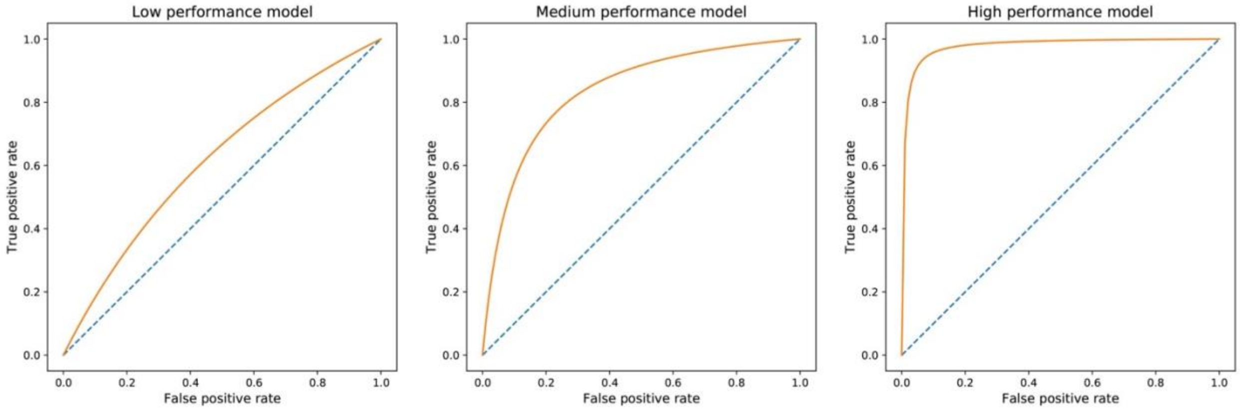


Illustration of possible ROC curves depending on the effectiveness of the model. On the left, the model has to sacrifice a lot of precision to get a high recall. On the right, the model is highly effective: it can reach a high recall while keeping a high precision.

Note: AUC=0.5 means the model does a random guess. AUC=0.3 (very likely) means your label might be wrong, it could be actual AUC=0.7 (VISA Sr DS interview as of 22/7).

Linear Regression Outliers

How to identify outliers?

- Standard residuals

How to deal with outliers?

- A measurement error or data entry error, correct the error if possible. If you can't fix it, remove that observation because you know it's incorrect.
- Not a part of the population you are studying (i.e., unusual properties or conditions), you can legitimately remove the outlier.
- A natural part of the population you are studying, you should not remove it.

- What you should do in that case is two separate analyses, one with the outlier in the analysis and one with the outlier removed from the analysis. Then report both analyses and let the reader make the decision as to which one should be used.

How to make outliers less influence to the linear regression?

- Robust regression uses a method called *iteratively reweighted least squares to assign a weight to each data point*. This method is less sensitive to large changes in small parts of the data. As a result, robust linear regression is less sensitive to outliers than standard linear regression.

Linear Regression Multicollinearity

How to detect multicollinearity?

1. *The Variance Inflation Factor (VIF)* quantifies how much the variance is inflated. The VIF for the j^{th} predictor is:

$$VIF_j = \frac{1}{1 - R_j^2}$$

where R_j^2 is the R^2-value obtained by regressing the j^{th} predictor on the remaining predictors (considering the x_j as the dependent variable and all others are independent variables). VIF is 1 means that there is no correlation among the j^{th} predictor and the remaining predictor variables, and hence the variance of b_j is not inflated at all. *The general rule of thumb is that VIFs exceeding 4 warrant further investigation, while VIFs exceeding 10 are signs of serious multicollinearity requiring correction.*

2. Calculate *the correlation coefficient matrix* for all the independent variables.

How to deal with multicollinearity?

One solution to dealing with multicollinearity is to remove some of the violating predictors from the model.

Linear Regression High Cardinality

What is the high cardinality?

A categorical feature is said to possess *high cardinality* when there are too many of these unique values. One-hot Encoding becomes a big problem in such a case since we have a separate column for each unique value in the categorical variable. This leads to two problems (1) storage space consumption; (2) *the curse of dimensionality* (当维数增大时，空间数据会变得更稀疏，这将导致 bias 和 variance 的增加，最后影响模型的预测效果。解决办法：增加 sample size 或者使用 dimensionality reduction).

How to deal the feature with a high cardinality?

A simple aggregation method. *Leave instances belonging to a value with high frequency as they are and replace the other instances with a new category which we will call other*.

Say our column color has 100 values and our threshold is 90% (that is 90). We have 5 different categories of colors: Red (50), Blue(40), Yellow (5), Green (3) and Orange (2). The numbers within the bracket indicate how many instances of that category are present in the column.

We see that Red (50) + Blue (40) reaches our threshold of 90. In that case, we retain only 2 categories (Red, Blue) and mark all other instances of other colors as the "*Other*". *Thus we have reduced cardinality from 5 to 3 (Red, Blue, Other).*

What is the curse of dimensionality and how to cure the "curse"?

- Increase in dimensionality *without* increasing the number of samples will increase the sparsity in the training instances, which makes it difficult for any learning method to produce reliable results.
- Increase in dimensionality also increase the "sparsity" of the data distribution and it makes the distance between samples are more identical.
- Add more training samples.
- Reduce the dimension of the data for instance using PCA.

Embedding vs. one-hot encoding

The one-hot encoding technique has two main drawbacks:

1. For high-cardinality variables — those with many unique categories — the dimensionality of the transformed vector becomes large (see previous high cardinality).
2. The mapping is completely uninformed: "similar" categories are not placed closer to each other in embedding space.

What is embedding? More details to be added…

An embedding is a mapping of a discrete — categorical — variable to a vector of continuous numbers.

https://towardsdatascience.com/neural-network-embeddings-explained-4d028e6f0526

Boosting Algorithms

What's the differences between AdaBoost and Gradient Boosting?

See Ace the Data Science Interview page 100.

Bagging/Boosting Variance and Bias

- Bagging algorithm (random forest) decreases ensemble variance by averaging, but has little effect on the ensemble bias, and solves the overfitting issues in a model.
- Boosting (AdaBoost/GBM) decreases the bias, not variance. It is because these mislabeled samples are with more weights, then the model would have less ensemble bias.

- Bagging algorithm (random forest) uses the bootstrapping – sampling a dataset w/ replacement. Boosting algorithm (AdaBoost/GBM/Xgboost) uses the full sample set for each iteration. Thus, they perform like sampling w/o replacement. While for Xgboost, the ratio of sampling of each tree can be modified with the following hyperparameter: subsample [default=1]

Cross Validation (Model Selection)

Version 1.

Cross-validation usually helps to avoid the need of a validation set.
The basic idea with training/validation/test data sets is as follows:

1. Training: You try out different types of models with different choices of hyperparameters on the training data (e.g. linear model with different selection of features, neural net with different choices of layers, random forest with different values of mtry).
2. Validation: You compare the performance of the models in Step 1 based on the validation set and select the winner. This helps to avoid wrong decisions taken by overfitting the training data set.
3. Test: You try out the winner model on the test data just to get a feeling how good it performs in reality. This unravels overfitting introduced in Step 2. Here, you would not take any further decision. It is just plain information.

Now, *in the case where you replace the validation step by cross-validation*, the attack on the data is done almost identically, but you only have a training and a test data set. *There is no need for a validation data set.*

1. Training: See above.
2. Validation: You do cross-validation on the training data to choose the best model of Step 1 with respect to cross-validation performance (here, the original training data is repeatedly split into a temporary training and validation set). The models calculated in cross-validation are only used for choosing the best model of Step 1, which are all computed on the full training set.
3. Test: See above.

Version 2. (From Andrew Ng and similar to Version 1. (a))

Suppose we want to select different models with different d values. The data set is separated into 2 sets – train/test data sets.
$d=1$, $h_\theta(x)=\theta_0+\theta_1 x$, train the model and get $\theta^{(1)}$ such that $J_{test}\left(\theta^{(1)}\right)$ is minimized.
$d=2$, $h_\theta(x)=\theta_0+\theta_1 x+\theta_2 x^2$, then where $\theta^{(2)} \rightarrow J_{test}\left(\theta^{(2)}\right)$
…
$d=10$, $h_\theta(x)=\theta_0+\theta_1 x+\theta_2 x^2+\cdots+\theta_{10} x^{10}$, then where $\theta^{(10)} \rightarrow J_{test}\left(\theta^{(10)}\right)$
Suppose d=5 is the optimal solution. Then how wells does the model generalizes? Report test set error $J_{test}\left(\theta^{(5)}\right)$.
Problem: $J_{test}\left(\theta^{(5)}\right)$ is likely to be an optimistic estimate of *generalization error*. i.e. our extra parameter (d=degree of polynomial) is fit to test set.

Divide the data set into 3 sets:

Training Set/Cross Validation Set (CV)/Test Set and the ratio is like 60%/20%/20%

Model Selection using Cross Validation
$d = 1$ $h_\theta(x) = \theta_0 + \theta_1 x$, then $\min_\theta J(\theta) \to \theta^{(1)} \to J_{CV}\left(\theta^{(1)}\right)$
$d = 2$ $h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2$, then $\min_\theta J(\theta) \to \theta^{(2)} \to J_{CV}\left(\theta^{(2)}\right)$
…
$d = 10$ $h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \cdots + \theta_{10} x^{10}$, then $\min_\theta J(\theta) \to \theta^{(10)} \to J_{CV}\left(\theta^{(10)}\right)$
Suppose $J_{CV}\left(\theta^{(4)}\right)$ has the smallest CV error and pick $d = 4$.
Estimate the generalization error for test set $J_{test}\left(\theta^{(4)}\right)$.

Optimizations

Gradient Descent (also called batch gradient descent)

$J_{train}(\theta) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$

```
Repeat {
        for every $j = 0, \dots, n$
                $\theta_j := \theta_j - \alpha \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)x_j^{(i)}$
}
```

Notes: where n is the number of features and m is number of samples. Suppose update K iterations, the total computation is $O(Kmn)$. When m is large, computing is very expensive to update one iteration.

Stochastic Gradient Descent

$J_{train}(\theta) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$

```
1. Randomly shuffle dataset
2. Repeat {
   For $i = 1, \dots, m$ { // GD on each observation, so the convergence path is the zigzag shape
           for every $j = 0, \dots, n$
                   $\theta_j := \theta_j - \alpha\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)x_j^{(i)}$
           }
   }
```

Suppose update K iterations, the total computation is $O(Kn)$. Each iteration only takes 1 observation, so the convergence path is a zigzag shape.

Mini-Batch Gradient Descent

$J_{train}(\theta) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$

Say $b = 10$ (batch size) and $m = 1000$

```
Repeat {
For $i = 1,11,21,\ldots,991$ {
        $$\theta_j := \theta_j - \alpha \frac{1}{10} \sum_{k=i}^{i+9} \left(h_\theta\left(x^{(k)}\right) - y^{(k)}\right) x_j^{(k)}$$
    (for every $j = 0,\ldots,n$)
    }
}
```

Note: It's between batch GD and stochastic GD and the vectorization can be parallelized.
Suppose update K iterations, the total computation is $O(Kbn)$.

RMSprop (Root Mean Square Propagation)

On iteration t:

Compute dW, db on current *mini-batch*

$S_{dW} = \beta_2 S_{dW} + (1-\beta_2) dW^2$

$S_{db} = \beta_2 S_{db} + (1-\beta_2) db^2$

$w := w - \alpha \frac{dW}{\sqrt{S_{dW}}+\epsilon}$

$b := b - \alpha \frac{db}{\sqrt{S_{db}}+\epsilon}$

Notes:

The algorithm damps out the oscillations in the gradient descent, so a larger learning rate can be applied to speed up the learning algorithm.

Adam (ADAptive Moment estimation)

$V_{dW} = 0, S_{dW} = 0, V_{db} = 0, S_{db} = 0$

On iteration t:

Compute

$V_{dW} = \beta_1 V_{dW} + (1-\beta_1) dW, V_{db} = \beta_1 V_{db} + (1-\beta_1) db$ // the "momentum" β_1

$S_{dW} = \beta_2 S_{dW} + (1-\beta_2) dW^2, S_{db} = \beta_2 S_{db} + (1-\beta_2) db^2$ // the "RMSProp" β_2

$V_{dW}^{corrected} = V_{dW}/(1-\beta_1^t), V_{db}^{corrected} = V_{db}/(1-\beta_1^t)$ // t is the iteration number

$S_{dW}^{corrected} = S_{dW}/(1-\beta_2^t), S_{db}^{corrected} = S_{db}/(1-\beta_2^t)$

$w := w - \alpha \frac{V_{dW}^{corrected}}{\sqrt{S_{dW}^{corrected}}+\epsilon}$

$b := b - \alpha \frac{V_{db}^{corrected}}{\sqrt{S_{db}^{corrected}}+\epsilon}$

Configuration parameters:

α: needs to be tuned

β_1: 0.9 // the first moment

β_2: 0.999 // the second moment

ϵ: 10E-8
Note:

- Adam combines the best properties of the AdaGrad (1st moment) & RMSProp (2nd moment) algorithms to provide an optimization algorithm that can handle sparse gradients on noisy problems.
- Adam is relatively easy to configure where default parameters do well on most cases.

Feature Selection in Machine Learning

Embedded Methods

They are used to optimize the objective function or performance if a learning algorithm or model. The intrinsic model building metric is used during learning.
Regularization: Regularization adds a penalty term to different parameters of the machine learning model for avoiding overfitting in the model. This penalty term is added to the coefficients; **hence it shrinks some coefficients to zero. Those features with zero coefficients can be removed from the dataset.** The types of regularization techniques are L1 Regularization (Lasso Regularization) or Elastic Nets (L1 and L2 regularization).

Filter Methods (on the basis of statistics measures)

The filter methods pick up the intrinsic properties of the features (i.e. the relevance of the features) measured via univariate statistics.
Chi-squared test is used for categorical features in a dataset. We calculate the Chi-square between *a categorical feature* and *a categorial target* and select the desired number of features with the best Chi-square scores. Note that the Fisher's exact test is for a small-sized samples.
Correlation (Pearson's correlation) is a measure of the linear relationship of *2 or more continuous variables*. If two variables are correlated, we can predict one from the other. Therefore, the model only really needs one of them.

Model explainability/interpretation

- Feature importance – Which features impact most model predictions
- Partial dependence plots – How is ONE feature impacting predictions
- SHAP values – How are ALL features impacting ONE prediction

Partial Dependent Plot

The partial dependent plot (PDP) shows the global effect that a single input feature has on the model prediction. The input feature of interest is set to a constant value for **all examples** in the training data. The model then scores this adjusted data and average over the training data to calculate a single point on the PDP. This process is repeated to calculate the PDP over the input feature range in the training data.

The original dataset is {obs1:[2,4,6], obs2:[1,3,5], obs3:[4,2,1]}. Consider the PDP on x2, the possible new dataset is {obs1:[2,2,6], obs2:[1,2,5], obs3:[4,2,1]} and then compute y_hat1, y_hat2, and y_hat3, then average y_hat's. Repeat the process for a different x2 value say x2=3 and so on.

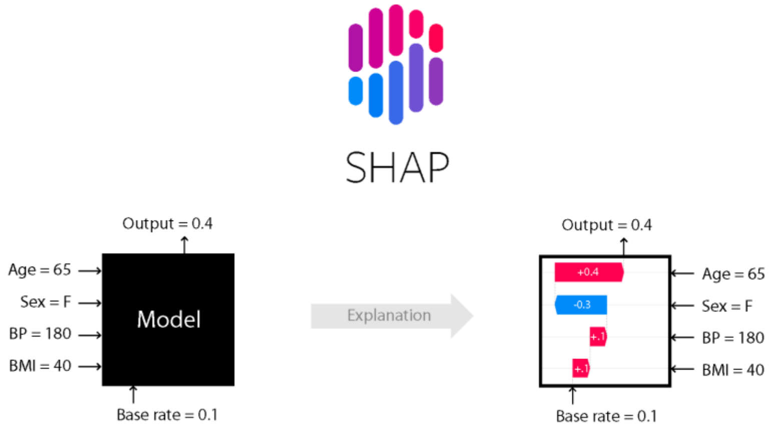
SHAP interpretability

Shapley Additive Explainability (SHAP) is another popular interpretability method to explain individual predictions (local/global). Shapley values are obtained by incorporating concepts from Cooperative Game Theory and local explanations (i.e. given a set of features, Cooperate Game Theory defines how to fairly distribute the attribution amongst all the features that are working in coordination).

The value of the i-th feature of j-th instance is its contribution to the prediction f(x_i), weighted and summed over all possible feature value combinations. An abstract intuition to understand this measure can be shown in simple linear model

$y = \beta_0 + \beta_1 x^1 + \cdots + \beta_m x^m$, the Shaply value of the i-th feature on the j-th sample is simply

$\phi(x_j^i) = \beta_i(x_j^i - \bar{x}^i)$



For example, for a particular feature X with a feature_value=0.39, mean_feature_value=-0.04 and shap_value=0.34. It means the feature_value=0.39 has a positive contribution shap_value=0.34 to the prediction.

XGBoost

Q: How does XGBoost do the parallel computation?

A: XGBoost doesn't run *multiple trees* in parallel, you need predictions after each tree to update gradients. Rather it does the parallelization *WITHIN* a single tree my using openMP to create branches independently.

To observe this,build a giant dataset and run with n_rounds=1. You will see all your cores firing on one tree. This is why it's so fast- well engineered.

Q: How does XGBoost treat the missing values?

A: (Answered by Tianqi Chen) Internally, XGBoost will automatically learn what is the best direction to go when a value is missing. Equivalently, this can be viewed as automatically "learn" what is the best imputation value for missing values based on reduction on training loss.

Q: How does XGBoost calculate the important score? What are the options?

A: Get feature importance of each feature. For tree model Importance type can be defined as:

- ‘weight’: the number of times a feature is used to split the data across all trees.
- ‘gain’: the average gain across all splits the feature is used in.
- ‘cover’: the average coverage across all splits the feature is used in.
- ‘total_gain’: the total gain across all splits the feature is used in.
- ‘total_cover’: the total coverage across all splits the feature is used in.

Light GBM

Light GBM (LGB) is a gradient boosting framework that uses a tree based learning algorithm that offer the following advantages over prior implementations of GBM’s:

- Faster training speed and higher efficiency: Light GBM uses a *histogram based algorithm* (i.e. it buckets continuous feature values into discrete bins which fasten the training procedure).
- Lower memory usage: Replaces continuous values to discrete bins which result in *lower memory usage*.
- Better accuracy than other boosting algorithm: It produces much more complex trees by following *a leaf-wise split approach (BFS)* rather than *a level-wise approach (DFS)*

which is the main factor in achieving higher accuracy. However, it can sometimes lead to overfitting which can be avoided by setting the max_depth & num_leaves parameter.

- Compatibility with Large Datasets: It is capable of performing equally good with large datasets with a significant reduction in training time as compared to XGBOOST.
- Parallel learning is supported.

Comparison of Two Means

The hypothesis is

$$H_0: \mu_1 = \mu_2$$
$$H_a: \mu_1 \neq \mu_2$$

Tests of Significance for *Two Unknown Means and Known Standard Deviations*

Given samples from two normal populations of size n_1 and n_2 with unknown means μ_1 and μ_2 and known standard deviations σ_1 and σ_2, the test statistic comparing the means is known as *the two-sample z statistic*

$$z = \frac{(\bar{x}_1 - \bar{x}_2) - (\bar{\mu}_1 - \bar{\mu}_2)}{\sqrt{\sigma_1^2/n_1 + \sigma_2^2/n_2}}$$

Which has the standard normal distribution $N(0,1)$.

Tests of Significance for *Two Unknown Means and Unknown Standard Deviations*

In general, the population standard deviations are not known, and are estimated by the calculated values s_1 and s_2. In this case, the test statistic is defined by *the two-sample t statistic*

$$t = \frac{(\bar{x}_1 - \bar{x}_2) - (\bar{\mu}_1 - \bar{\mu}_2)}{\sqrt{s_1^2/n_1 + s_2^2/n_2}}$$

Although the two-sample statistic does not exactly follow the t distribution (since two standard deviations are estimated in the statistic), conservative p-values may be obtained using the $t(k)$ distribution where k represents the smaller of $n_1 - 1$ and $n_2 - 1$.

P-value

What is the p-value?

- A p-value is a statistical measurement used to validate a hypothesis against observed data.
- The lower the p-value, the greater the statistical significance of the observed difference.
- A p-value of 0.05 or lower is generally considered statistically significant.
- A p-value measures the probability of obtaining the observed results, assuming that the null hypothesis is true.
- P-value can serve as an alternative to or in addition to preselected confidence levels for hypothesis testing.

A/B Testing

What's the A/B testing?

A/B testing is to compare 2 or more versions.

What's the null hypothesis and the alternative hypothesis?

The null hypothesis is that the change in the design made for the test group **would results in no change in the conversion rate.** The alternative hypothesis is the opposing that the change in the design for the test group **would results in an improvement (or reduction) in the conversion rate.**

- $p \leq 0.05$ (*Note that the equal sign means 'significant'*)
  - Reject H0 -> Change in the conversion rate -> Significant test result
- $p > 0.05$
  - Fail to reject H0 -> No change in the conversion rate -> Not significant test result

How to determine the minimum sample size?

You will need *the baseline conversion rate* and *the minimum detectable effect (MDE)/minimum effect of interest (MEI)*, which is the minimum difference between the control and test group that you or your team will determine to be worth the investment of making the design change in the first place. The equation for minimum sample size in each group (assumes equal sized groups) is

$$n = \frac{2\big(\bar{p}(1-\bar{p})\big)\big(Z_\beta - Z_{\alpha/2}\big)^2}{(p_B - p_A)^2}$$

Z_β: z-score represents the desired Power (typically 0.84 for 80% power – see below for explanation of the Power)

$Z_{\alpha/2}$: z-score represents the desired level of statistical significance (typically 1.96)

$\bar{p}$: pooled probability or the average of p_A and p_B

p_A: success rate of control group

p_B: success rate of test group

What is the Power?

Power is also known as $1-\beta$, it can be explained as the strength of your test to detect an actual difference in your variant. Conversely, β is the probability that your test does not reject the null hypothesis when it should actually be rejecting the null hypothesis. *The higher the Power, the lower the probability of a Type II error.* Experiments are usually set at a power level of 80%, or $\beta = 20\%$. Another way of putting it is: if there is a difference in the test, you're willing to make Type II error 20% of the time.

How to deal with small sample sizes?

We can lower the required sample size by:

- Reducing the confidence threshold / increasing the significance threshold – accepting higher risk of false positives
- Increasing the minimum effect of interest (MEI/MDE)
- By accepting the test will have lower power against a particular MEI

Can you run an A/B test with unequal sample sizes? (Hyperlink)

TL;DR: Yes, but you wouldn’t want to.

You can run an experiment with an unequal allocation (e.g. 10–90) as long as you don’t modify the allocation while the experiment is running. However it will be less efficient than a 50–50 allocation – either your test will have less power, or you will need to run it longer to achieve a comparable result.

例如原计划两周的 AB testing，一周后发现 p-value<0.05 了， 这时候能不能结束？怎么从统计上解释不能提前结束？

答案：一般做 ab test 都是根据想要达到的 statistical power，一般是 0.8，来计算一个 sample size，然后用 sample size 来预计需要做 testing 的时间。在你的 case 里面算出来的时间是两周，如果你只做了一周，预设的 sample size 没达到，那么 power 就会偏低，导致 type 2 error 太高，实验结果不可信。*p-value 小于 0.05 只能保证 type 1 error 比较小，但 sample size 没达到会导致 type 2 error 太高。*我们做 ab test 要两种 error 都低实验结果才可信。

An example: Ads Click Through Rate (CLR) (Hyperlink)

$$\bar{X}_1 \sim N(p_1, p_1(1-p_1)/n_1)$$

$$\bar{X}_2 \sim N(p_2, p_2(1-p_2)/n_2)$$

$$\bar{X}_1 - \bar{X}_2 \sim N\left(p_1 - p_2, \frac{p_1(1-p_1)}{n_1} + \frac{p_2(1-p_2)}{n_2}\right)$$

Hypothesis:

$$H_0: \bar{X}_1 - \bar{X}_2 = 0, H_a: \bar{X}_1 - \bar{X}_2 \neq 0$$

t-test statistic:

$$t = \frac{\bar{X}_1 - \bar{X}_2 - 0}{SE} = \frac{\bar{X}_1 - \bar{X}_2 - 0}{\sqrt{\frac{p_1(1-p_1)}{n_1} + \frac{p_2(1-p_2)}{n_2}}}$$

Since standard deviation of the sample $S_1 = \sqrt{p_1(1-p_1)/n_1}$ is different from $S_2 = \sqrt{p_2(1-p_2)/n_2}$, we use the unpooled standard error t-test (*usually one SE should twice more than another SE, but in this case they're not that different*). For unpooled t-test, the degree of freedom we choose $\min(n_1 - 1, n_2 - 1)$.

Causal Inference

How to discover causal inference?

Randomized Controlled Tests (or A/B test)

An example and steps

Goal: Does sending promotion emails increase customer’s purchase conversion?

The RCT consists of 5 steps

1. Select participants (randomly)
2. Split them into 2 groups (randomly)
3. Participants in the treatment group get emails & participants in the control group gets no emails
4. Monitor purchase conversion over time
5. Make decision & draw conclusion: sending emails “causes” purchase conversion increase

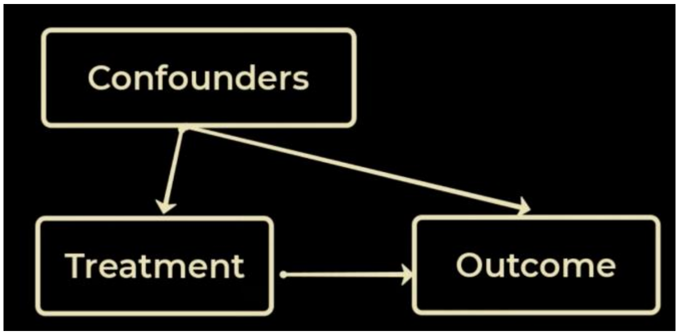
Challenges & Concepts

1. Confounders
   Some factors, rather than the control variables, impact the outcome as well. The randomization is important in selection and split the groups.
2. Selection bias
   The selected group isn’t a good representative for the whole population.
3. Counterfactuals
   A sample in the control group also outputs an outcome with the treatment, but the outcome is imputed using machine learning. Control 组中没有接受 Treatment 的也产生一个 outcome，但是这个 outcome 是人为的。

Causal inference assumptions

1. Causal Markov Condition (Markov assumption)

- Causal graph
- Directed Acyclic Graph (DAG)



2. SUTVA (Stable Unit Treatment Value Assumption)
   Control and Treatment groups do not influence each other
3. Igonorability
   No unknown confounders.

How to measure the causal inference?

Average Treatment Effect (ATE) and conditional ATE.

Central limit theorem

In probability theory, the central limit theorem (CLT) establishes that, in many situations, when independent random variables are summed up, their properly normalized sum tends toward a normal distribution even if the original variables themselves are not normally distributed.

If $X_1, X_2, \dots, X_n, \dots$ are random samples drawn from a population with overall mean μ and finite variance σ^2, and if $\bar{X}_n$ is the sample mean of first n samples, then the limiting form of the distribution, $Z = \lim_{n\to\infty}\left(\frac{\bar{X}_n-\mu}{\frac{\sigma}{\sqrt{n}}}\right)$, is a standard normal distribution.

Law of large numbers

In probability theory, the law of large numbers (LLN) is a theorem that describes the result of performing the same experiment a large number of times. According to the law, the average of the results obtained from a large number of trials should be close to the expected value and tends to become closer to the expected value as more trials are performed.

It is also important to note that the LLN only applies to the average. Therefore, while

$$\lim_{n\to\infty}\sum_{i=1}^{n}\frac{X_i}{n} = \bar{X}$$

Bayes' Theorem

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

where A and B are events and $P(B) \neq 0$.

What are gradient vanishing/exploding in deep neural networks

Vanishing

As the backpropagation algorithm advances downwards (or backward) from the output layer towards the input layer, the gradients often get smaller and smaller and approach zero which eventually leaves the weights of the initial or lower layers nearly unchanged. As a result, the gradient descent never converges to the optimum. This is known as the vanishing gradients problem.

Exploding

On the contrary, in some cases, the gradients keep on getting larger and larger as the backpropagation algorithm progresses. This, in turn, causes very large weight updates and causes the gradient descent to diverge. This is known as the exploding gradients problem.

Why it happens?

Certain activation functions, like the logistic function (sigmoid), have a very huge difference between the variance of their inputs and the outputs. In simpler words, they shrink and transform a larger input space into a smaller output space that lies between the range of [0,1]. Observing the above graph […missing…] of the sigmoid function, we can see that for larger inputs (negative or positive), its output saturates at 0 or 1 […but…] with *a derivative* very close to zero. Thus, when the backpropagation algorithm chips in, it virtually has no gradients to propagate backward in the network, and whatever little residual gradients exist keeps on diluting as the algorithm progresses down through the top layers. So, this leaves nothing for the lower layers.

What are the solutions?

1. Weight initializations

Randomly initialize the connection weights for each layer in the network as described in the following equation which is popularly known as Xavier initialization (after the author's first name) or Glorot initialization (after his last name).
Weights follows a normal distribution with mean 0 and variance $\sigma^2 = 1/n_{avg}$ where $n_{avg} = (n_{input} + n_{output})/2$ and n_{input} and n_{output} are numbers of layer input and number of neuron for that layer.

2. Using Non-saturating Activation Functions (this one is for the vanishing of gradients) such as ReLU (Rectified Linear Unit)

$$R(z) = \max(0, z)$$

3. Gradient Clipping

Another popular technique to mitigate the exploding gradients problem is to clip the gradients during backpropagation so that they never exceed some threshold. This is called Gradient Clipping. *This optimizer will clip every component of the gradient vector to a value between -1.0 and 1.0.* Meaning, all the partial derivatives of the loss w.r.t each trainable parameter will be clipped between -1.0 and 1.0

Batch Normalization

https://deepai.org/machine-learning-glossary-and-terms/batch-normalization

$$\mu = \frac{1}{m}\sum z^{(i)},\ \sigma^2 = \frac{1}{m}\sum\left(z^{(i)} - \mu\right)^2,\ z_{norm}^{(i)} = \frac{z^{(i)}-\mu}{\sqrt{\sigma^2+\epsilon}} \text{ and } \tilde{z}^{(i)} = \gamma z_{norm}^{(i)} + \beta$$

where γ and β are learnable parameters and $z^{(i)}$ means $z^{[l](i)}$ for lth layer. We are mapping from $z_{norm}^{(i)}$ to $\tilde{z}^{(i)}$ because we don't want to the hidden units always have zero mean and one variance.

Special case is when $\gamma = \sqrt{\sigma^2 + \epsilon}$ and $\beta = \mu$, then $\tilde{z}^{(i)} = z^{(i)}$.

$$x \xrightarrow{w^{[1]},b^{[1]}} z^{[1]} \xrightarrow{\beta^{[1]},\gamma^{[1]}\ BN} \tilde{z}^{[1]} \to a^{[1]} = g^{[1]}\left(\tilde{z}^{[1]}\right) \xrightarrow{w^{[2]},b^{[2]}} z^{[2]} \xrightarrow{\beta^{[2]},\gamma^{[2]}} \tilde{z}^{[2]} \to \cdots$$

Parameters are $w^{[1]}, b^{[1]}, \beta^{[1]}, \gamma^{[1]}, \dots, w^{[l]}, b^{[l]}, \beta^{[l]}, \gamma^{[l]}$ and $b^{[t]}$ are actually all zeros.

BN works within a mini-batch

- For the t^{th} mini-batch

$$X^{\{t\}} \xrightarrow{w^{[1]},b^{[1]}} z^{[1]} \xrightarrow{\beta^{[1]},\gamma^{[1]}\ BN} \tilde{z}^{[1]} \to a^{[1]} = g^{[1]}\left(\tilde{z}^{[1]}\right) \xrightarrow{w^{[2]},b^{[2]}} z^{[2]} \xrightarrow{\beta^{[2]},\gamma^{[2]}} \tilde{z}^{[2]} \to \cdots$$

- Compute forward prop on $X^{\{t\}}$
- In each hidden layer, use BN to replace $z^{[1]}$ with $\tilde{z}^{[l]}$
- Use back prop to compute gradient $dw^{[l]}$, ~~$db^{[l]}$~~, $d\beta^{[l]}, d\gamma^{[l]}$
- Update the parameters $w^{[l]} \coloneqq w^{[l]} + a\ dw^{[l]}, \dots$

Why BN works?

- Normalized the inputs for the hidden layers (similar idea as to normalize the inputs for the model)
- Covariate shift. Consider a mapping $x \to y$, when x changes, the mapping needs to be retrained. The batch normalization reduces the covariate shift *for each hidden layer* – even small changes in the input won’t affect the layer output too much.
- Batch norm as regularization. For each mini-batch, the mapping $z^{[l]} \to \tilde{z}^{[l]}$ within that mini batch add some noise. Similar to dropout, this has a *slight* regularization effect.

BN at test time

At the training time, for each mini-batch $\{t\}$, to keep records for l layer, $\mu^{\{1\}[l]}, \mu^{\{2\}[l]}, \mu^{\{3\}[l]}, \dots$ and compute μ (exponentially weighted average) likewise for σ^2 at l layer, $\sigma^{2\{1\}[l]}, \sigma^{2\{2\}[l]}, \sigma^{2\{3\}[l]}, \dots$ and compute σ^2. Then

$$z_{norm}^{(i)} = \frac{\left(z^{(i)} - \mu\right)}{\sqrt{\sigma^2 + \epsilon}}$$

$$\tilde{z}^{(i)} = \gamma\ z_{norm}^{(i)} + \beta$$

Imbalanced Dataset

https://github.com/scikit-learn-contrib/imbalanced-learn

All these approaches aim at rebalancing (partially or fully) the dataset:

- Undersampling consists in sampling from the majority class in order to keep only a part of these points
    - Random undersampling: Randomly remove samples from the majority class, with or without replacement. This is one of the earliest techniques used to alleviate imbalance in the dataset, however, it *may increase the variance of the classifier and is very likely to discard useful or important samples*.
    - Cluster: Cluster centroids is a method that replaces cluster of samples by the cluster centroid of a K-means algorithm, where the number of clusters is set by the level of undersampling.
    - Disadvantages: (1) It can discard useful information about the data itself which could be necessary for building rule-based classifiers such as Random Forests. (2) The sample chosen by random undersampling may be a biased sample. And it will not be an accurate

representation of the population in that case. Therefore, it can cause the classifier to perform poorly on real unseen data.

- Oversampling consists in replicating some points from the minority class in order to increase its cardinality
  - Random Oversampling: Random Oversampling involves supplementing the training data with multiple copies of some of the minority classes. Oversampling can be done more than once (2x, 3x, 5x, 10x, etc.) This is one of the earliest proposed methods, that is also proven to be robust. Instead of duplicating every sample in the minority class, *some of them may be randomly chosen with replacement*.
  - Generating synthetic data consists in creating new synthetic points from the minority class (see SMOTE method for example) to increase its cardinality.
  - Disadvantages: (1) It increases the likelihood of overfitting since it replicates the minority class events. (2) While generating synthetic examples, SMOTE does not take into consideration neighboring examples can be from other classes. This can increase the overlapping of classes and can introduce additional noise. (3) SMOTE is not very practical for high dimensional data.
- It's possible to combine oversampling and undersampling techniques into a hybrid strategy. Common examples include SMOTE and Tomek links or SMOTE and Edited Nearest Neighbors (ENN).

SMOTE

SMOTE (synthetic minority oversampling technique) is one of the most commonly used oversampling methods to solve the imbalance problem. It aims to balance class distribution by randomly increasing minority class examples by replicating them. SMOTE synthesizes new minority instances between existing minority instances. It generates the **virtual training records by linear interpolation** for the minority class.

- **Step 1:** Setting the minority class set $\boldsymbol{A}$, for each $\boldsymbol{x} \in \boldsymbol{A}$**,** the **$\boldsymbol{k}$-nearest neighbors of** x are obtained by calculating the **Euclidean distance** between $\boldsymbol{x}$ and every other sample in set $\boldsymbol{A}$.
- **Step 2:** The sampling rate $\boldsymbol{N}$ is set according to the imbalanced proportion. For each $x_l \in A$, $\boldsymbol{N}$ examples (i.e $x_{l,1}, x_{l,2}, \dots, x_{l,n}$) are *randomly selected* from its k-nearest neighbors, and they construct the set A_l^1. For example if $n = 3$ and $k = 5$, 3 of the 5-nearest neighbors of x_l are randomly selected and n can be either less or greater than k.
- **Step 3:** For each example $x_{l,i} \in A_l^1$ $(i = 1,2, \dots, N)$, the following formula is used to generate a new example

  $$x'_{l,i} = x_l + u \cdot |x_l - x_{l,i}|$$

  in which u follows a uniform distribution between 0 and 1.

K-mean

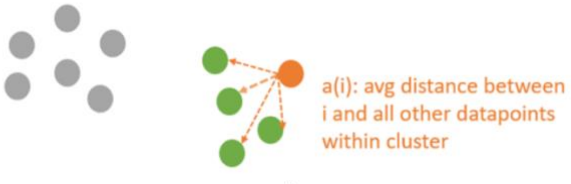
Details of implementation of the algorithm is saved in KMeans.py

NOTE: The Silhouette Method is used in combination with the Elbow Method for a more confident decision.

Silhouette coefficient for a particular data point i

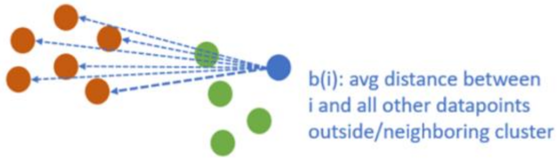
$$S(i) = \frac{b(i) - a(i)}{\max\big(a(i), b(i)\big)}$$

$a(i)$ is the average distance between i and all the other data points in the cluster which i belongs to.



a(i)

$b(i)$ is the average distance from i to all clusters which i does not belong to.


b(i): avg distance between
i and all other datapoints
outside/neighboring cluster

b(i)

For a value of K, to calculate the average Silhouette coefficient

$$AverageSilhouette = mean\{S(i)\}$$

Points to remember while calculating the Silhouette coefficient:

- The value of the silhouette coefficient is between $[-1, 1]$.
- A score of 1 denotes the best meaning that the data point i is very compact within the cluster to which it belongs and far away from the other clusters.
- The worst value is -1. Values near 0 denote overlapping clusters.

Generative Model vs. Discriminative Model
A Generative Model learns the joint probability distribution $P(X, Y)$. It predicts the conditional probability with the help of Bayes Theorem. *A Discriminative Model* learns the conditional probability distribution $P(Y|X)$.

Generative models use probability estimates and likelihood to model data points and distinguish between different class labels in a dataset. These models *are capable of generating new data instances*. However, they also have a major drawback. The presence of outliers affects these models to a significant extent.

Discriminative models, also called *conditional models*, tend to learn *the boundary between classes/labels* in a dataset. Discriminative models are *not capable of generating new data instances*.

In Math, training classifiers involve estimating f: X -> Y or P(Y|X)

Generative classifiers

- Assume some functional form for $P(Y)$, $P(X|Y)$
- Estimate parameters of $P(X|Y)$ and $P(Y)$ directly from training data
- Use Bayes rule to calculate $P(Y|X) = P(Y)P(X|Y)/P(X) = P(X, Y)/P(X)$

Discriminative Classifiers

- Assume some functional form for $P(Y|X)$
- Estimate parameters of $P(Y|X)$ directly from training data

Generative Classifiers

- Naïve Bayes (Bayesian networks)
- Hidden Markov Models (HMM)

Discriminative Classifiers

- Logistic regression
- Support Vector Machine
- Decision Tree
- Random Forest

Dense & Sparse Features
Dense features incorporate information from users/items pairs, historical statistics, predictions from upstream models [1] and etc. Typically *sparse features* include IDs of users/items, demographics, keywords and etc. Many recommendation models rely on both dense and sparse features in conjunction in order to achieve the best results. Dense features cannot completely replace sparse features and vice versa.

Dense and sparse features have different properties. For example, *the relative ordering of dense feature values are meaningful but this is rarer for sparse features*. In an online production system values for *sparse features are often restricted within a predefined dictionary while ranges of dense features can be undetermined*. Due to the inherent differences of dense and sparse features, they are processed in different ways.

Machine Learning System Design

- Search Ranking
- Feed Based System
- Recommendation System
- Self-Driving Car: Image Segmentation
- Entity Linking System
- Ad Prediction System

CNN

The pooling operation involves sliding a two-dimensional filter over each channel of feature map and summarizing the features lying within the region covered by the filter. For a feature map having dimensions $n_h \times n_w \times n_c$, the dimensions of output obtained after a pooling layer is

$$(n_h - f + 1)/s \times (n_w - f + 1)/s \times n_c$$

- n_h - height of feature map
- n_w - width of feature map
- n_c - number of channels in the feature map
- f - size of filter
- s - stride length

A common CNN model architecture is to have **a number of convolution and pooling layers stacked one after the other**.

Why to use Pooling Layers?

- Pooling layers are used to **reduce the dimensions of the feature maps**. Thus, it reduces the number of parameters to learn and the amount of computation performed in the network.
- The pooling layer **summarizes the features present in a region of the feature map generated by a convolution layer.** So, further operations are performed on summarized features instead of precisely positioned features generated by the convolution layer. This makes the model more robust to variations in the position of the features in the input image.

Type of pooling layers:

- Max pooling
- Average pooling
- Global pooling – global max pooling or global average pooling
  Global pooling reduces each channel in the feature map to a single value. Thus, an $n_h \times n_w \times n_c$ feature map is reduced to $1 \times 1 \times n_c$ feature map. This is equivalent to using a filter of dimensions $n_h \times n_w$ i.e. the dimensions of the feature map. Further, it can be either global max pooling or global average pooling.

The motivation of padding

- For a gray scale $(n \times n)$ image and $(f \times f)$ filter/kernel, the dimensions of the image resulting from a convolution operation is $(n - f + 1) \times (n - f + 1)$. **Thus, the image shrinks every time a convolution operation is performed.** This places an upper limit to the number of times such an operation could be performed before the image reduces to nothing thereby **precluding (阻止 v.) us from building deeper networks**.
- Also, the pixels on the corners and the edges are used much less than those in the middle.

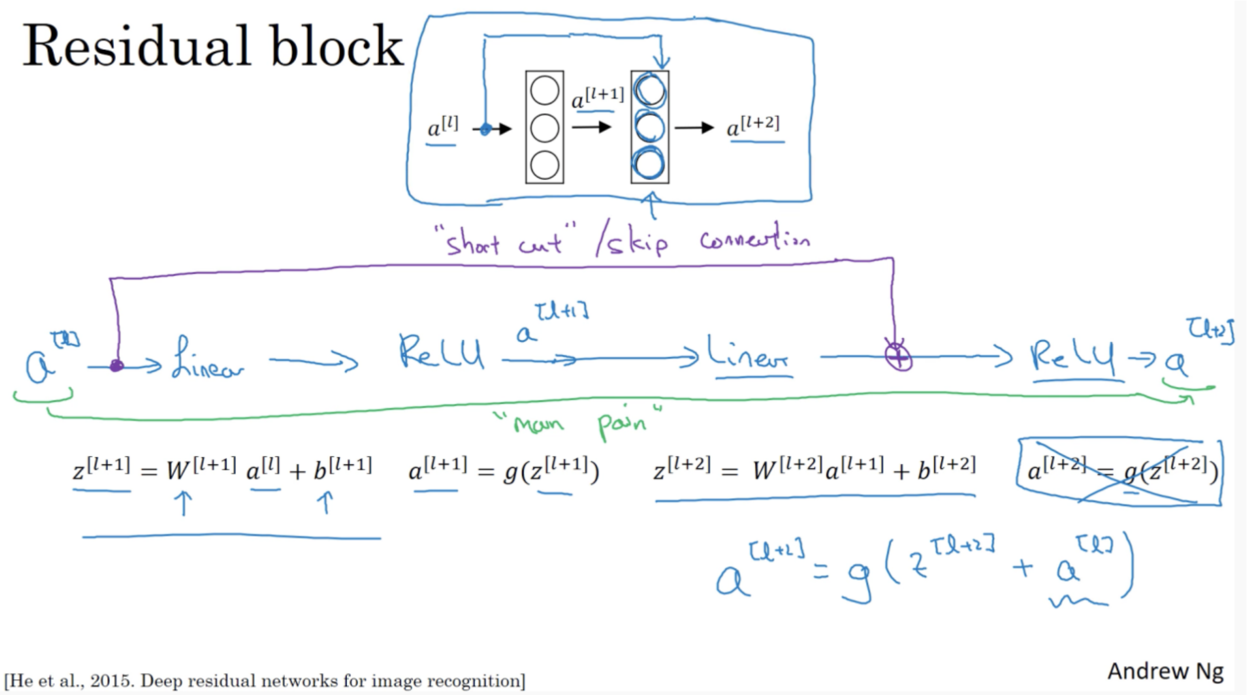
The same padding:

Same Padding: In this case, we add ‘p’ padding layers (p layers of zeros) such that the output image has the same dimensions as the input image.

$$[(n + 2p) \times (n + 2p) \text{ image}] * [(f \times f) \text{ filter}] \text{ -> } [(n \times n) \text{ image}]$$

where * represents a convolution operation
which gives p = (f – 1) / 2 (because n + 2p – f + 1 = n).

Residual Networks (ResNets):



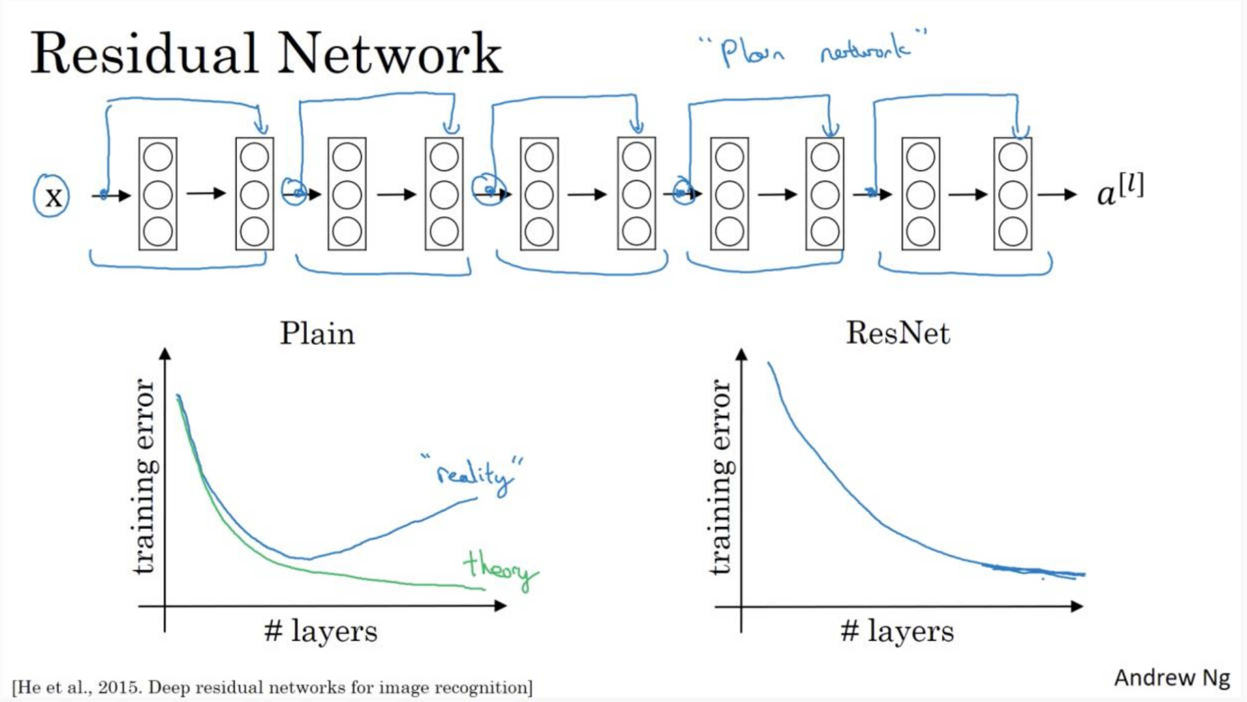
Residual block

$z^{[l+1]} = W^{[l+1]} a^{[l]} + b^{[l+1]}$ $\quad a^{[l+1]} = g(z^{[l+1]})$ $\quad z^{[l+2]} = W^{[l+2]} a^{[l+1]} + b^{[l+2]}$ $\quad \boxed{\cancel{a^{[l+2]} = g(z^{[l+2]})}}$

$a^{[l+2]} = g(z^{[l+2]} + a^{[l]})$

[He et al., 2015. Deep residual networks for image recognition]

Andrew Ng



Residual Network

[He et al., 2015. Deep residual networks for image recognition]

Andrew Ng

Why ResNets work?

- **The identity function** is easy for residual block to learn. It won’t hurt the overall performance as the depth increases. Deep networks hard to learn even the identity function.

A bag has 18 black balls and 9 white balls. On each turn you randomly select two balls and set them aside. If both balls are the same color you then add a black ball, otherwise you add a white ball. What is the probability that the last ball left in the bag is white? What if we instead start with 18 black balls and 36 white balls?

B(t) and W(t) are number of black balls and white balls at t
So we have B(0) = 18 and W(0) = 9 and also each time number of balls in the bag decreased by 1
So B(t+1) + W(t+1) = B(t) + W(t) – 1 and at the 26th step, B(26) + W(26) = 1
If we have BB, then B(t+1) = B(t) – 1 and W(t+1) = W(t)
If we have BW, then B(t+1) = B(t) – 1 and W(t+1) = W(t)
If we have WW, then B(t+1) = B(t) + 1 and W(t+1) = W(t) – 2
Notice that W is *always* odd because W(0) = 9 and it is decreased by 2 and B(26) + W(26) = 1. So the last one must be a white ball.

If there are 18 blacks balls and 36 white balls.
Because W(0) = 36, so W is *always* even and B(53) + W(53) = 1. So the last one must be a black ball.

You know that the probability it rains on any given day is 0.4. You are wondering if you should bring your umbrella with you today. You ask your friend if it’s raining right now and he says yes. Knowing that your friend may be lying to you with probability 0.2, what is the true probability that it is raining now?

R standards for prob of raining and Y stands for friends says yes.
So the question is asking for $P(R|Y)$ and we know $P(R) = 0.4$ and $P(Y|NR) = 0.2$.

$$P(R|Y) = \frac{P(Y|R)P(R)}{P(Y|R)P(R) + P(Y|NR)P(NR)} = \frac{0.8 \times 0.4}{0.8 \times 0.4 + 0.2 \times 0.6} = \frac{8}{11}$$