TABLE OF CONTENTS:

Please use search (Ctrl + F) / left side pane to find the below topics:

1. Linear Regression:

Q) What is a hypothesis function?

Ans) Hypothesis function is the approximated relationship between a dependent variable and the independent variables.

We are trying to fit our hypothesis function which can approximately estimate the dependent variable with respect to the independent variables.

Linear Regression relationship

$\mathbf{Y \approx \Theta_o + \Theta_1 X_1 + ... + \Theta_p X_p}$ *is the hypothesis function.*

Where $\mathbf{\Theta_o, \Theta_1, ..., \Theta_p}$ *are the coefficients of the linear regression equation.*

$$\mathbf{Y = \Theta_o + \Theta_1 X_1 + ... + \Theta_p X_p + \epsilon}$$

ϵ **is the error term associated with the predictions.**

$$\mathbf{\hat{Y} = \Theta_o + \Theta_1 X_1 + ... + \Theta_p X_p}$$

$\mathbf{\hat{Y}}$ **is the predicted vector of** $\mathbf{Y}$

$\mathbf{Y - \hat{Y} = \epsilon}$ **(Residual)**

$$\mathbf{f(X) = \Theta_o + \Theta_1 X_1 + ... + \Theta_p X_p}$$

$$\mathbf{Y = f(X) + \epsilon}$$

$$\mathbf{\hat{Y} = \hat{f}(X)}$$

Where $\mathbf{\hat{f}}$ **is the estimate for** $\mathbf{f}$**.**

Q) What are we trying to do?

Ans) We minimize the squared error to find out regression coefficients so as to fit f(X) as accurately as possible.

$$\begin{aligned}\mathbf{E(Y - \hat{Y})^2} &= \mathbf{E(f(X) + \epsilon - \hat{f}(X))^2} \\ &= \mathbf{[f(X) - \hat{f}(X)]^2 + Var(\epsilon)} \\ &= \mathbf{Reducible} \quad \mathbf{+ Irreducible}\end{aligned}$$

Reducible error can be of two kinds. Error due to variance and error due to bias.

$$\begin{aligned} \mathbf{E}(\mathbf{Y}-\hat{\mathbf{Y}})^{\mathbf{2}} &= \mathbf{Reducible} + \mathbf{Irreducible} \\ &= \mathbf{Var}(\hat{\mathbf{f}}(\mathbf{X})) + [\mathbf{Bias}(\hat{\mathbf{f}}(\mathbf{X}))]^{\mathbf{2}} + \mathbf{Var}(\epsilon) \end{aligned}$$

Find the coefficients:

$$\mathbf{e_i} = \mathbf{y_i} - \hat{\mathbf{y}}_\mathbf{i}$$

$$\mathbf{RSS} = \sum_{\mathbf{1}}^{\mathbf{m}} \mathbf{e_i^2} = \sum_{\mathbf{1}}^{\mathbf{m}} (\mathbf{y_i} - \boldsymbol{\Theta}_\mathbf{0} - \boldsymbol{\Theta}_\mathbf{1}\mathbf{x})^{\mathbf{2}}$$

$$\mathbf{By\ solving}\ \frac{\partial \mathbf{RSS}}{\partial \boldsymbol{\Theta}_\mathbf{0}} = \mathbf{0}\ \mathbf{and}\ \frac{\partial \mathbf{RSS}}{\partial \boldsymbol{\Theta}_\mathbf{1}} = \mathbf{0}$$

$$\mathbf{we\ can\ show\ that}$$

$$\hat{\boldsymbol{\Theta}}_\mathbf{1} = \frac{\sum_{\mathbf{1}}^{\mathbf{m}} (\mathbf{x_i} - \bar{\mathbf{x}})(\mathbf{y_i} - \bar{\mathbf{y}})}{\sum_{\mathbf{1}}^{\mathbf{m}} (\mathbf{x_i} - \bar{\mathbf{x}})^{\mathbf{2}}}$$

$$\hat{\boldsymbol{\Theta}}_\mathbf{0} = \bar{\mathbf{y}} - \hat{\boldsymbol{\Theta}}_\mathbf{1}\bar{\mathbf{x}}$$

Regression coefficients can also be obtained by an optimization algorithm called **Gradient Descent**. Gradient Descent algorithm uses the same squared error function but in a different way to arrive at the coefficients.

$$\mathbf{cost\ function}\ \mathbf{J}(\boldsymbol{\Theta}_\mathbf{0}, \boldsymbol{\Theta}_\mathbf{1}) = \frac{\mathbf{1}}{\mathbf{2m}} \sum_{\mathbf{1}}^{\mathbf{m}} (\boldsymbol{\Theta}_\mathbf{0} + \boldsymbol{\Theta}_\mathbf{1}\mathbf{x_i} - \mathbf{y_i})^{\mathbf{2}}$$

$$\mathbf{Where\ m\ is\ the\ number\ of\ observations}$$

This cost function corresponds to the batch gradient descent, where at each iteration cost function is calculated using all the observations present in the training set. Depending on the size of the training set, if large, m can be decreased to a fraction (**mini batch gradient descent**) or to 1 observation at a time (**stochastic gradient descent or online learning**) for the algorithm to converge.

Will this always converge?: Notice that the cost function for linear regression is always in the form of a quadratic function. Hence, it is a **convex function** and so the algorithm will always **converge** to a global minimum, unlike other complex models where gradient descent algorithm can be stuck at local minimum.

Choose the right learning rate: However, choosing the right value of learning rate is critical. If the learning rate is too small, gradient descent can be too slow to converge. While if it is too large, gradient descent can overshoot the minimum and may fail to converge or even diverge

Read more about Linear Regression: All about Linear Regression. This blog post is intended to the… | by Supreeth Manyam | Medium

2. Assumptions for Linear Regression? (LINER)

https://www.youtube.com/watch?v=0MFpOQRY0rw

a) (L) Linearity
b) (N) Normal Errors
c) No multicollinearity between independent variables (the Xs)
d) (I) Independent Error Terms: No autocorrelation in error
e) (E) Equal/Constant Error Variance: Residuals/errors have same variance (Homoscedasticity)
f) Exogeneity
g) Random - Data has no bias

If these assumptions are not followed, the coefficients or Standard Error values would not be reliable for Hypothesis Testing.

a) Linearity: Linear Relationship between the independent and dependent variables. The dependent variable = Linear Combination (Independent Variables)

Relationship between the predictors (X) and the response variable (Y) is **additive** and **linear**.

Additive: The effect of changes in a predictor X on response Y is independent of the values of the other predictors.
To handle the additive assumption, synergy or interaction between the variables can be introduced which can capture the fit of the response variable better. It is recommended

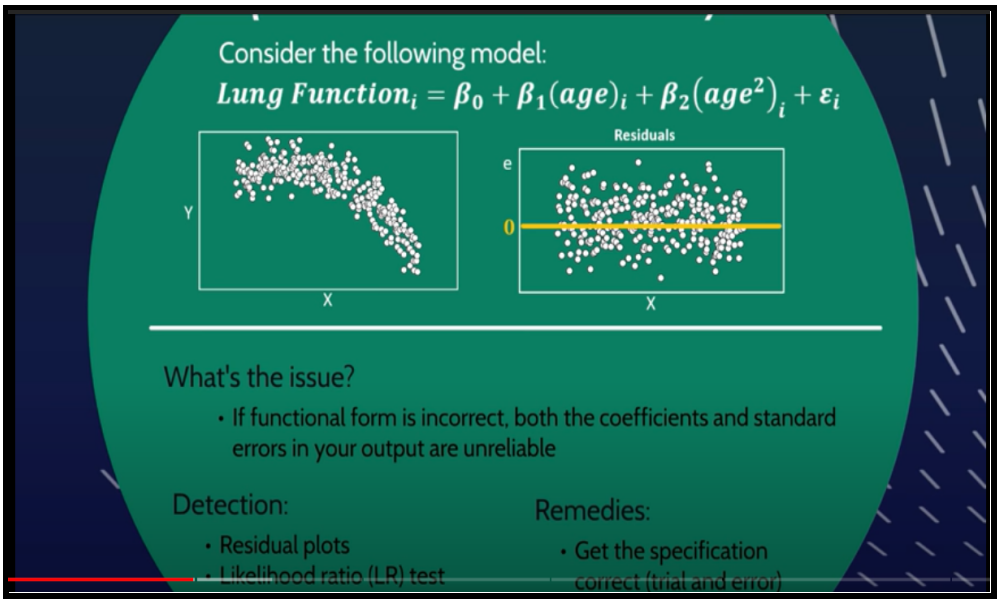
to include the variables X1 and X2, even if their interaction (X1*X2 or $X1^2$ or $X2^2$) is introduced.
The concept of interaction is also applicable to qualitative variables or a combination of qualitative and quantitative variables.

Linear: The change in Y due to unit change in X is constant.
Strong patterns in residual plots indicate non-linearity in the data.

Note: Even a quadratic/polynomial equation is a linear equation.

Detection: Residual plots, Likelihood-Ratio(LR) Test

Remedies: Trial and Error



b) Normal Errors: The **residuals** (error) which explain the variation in Y is **normally distributed (with mean = 0, constant variance (point e))** . The Gaussian (normal) seems like a good choice, because our errors look like they're symmetric about where the line would be, and that small errors are more likely than large errors.
But, as sample sizes increase, the normality assumption for the residuals is not needed. More precisely, if we consider repeated sampling from our population, for large sample sizes, the distribution (across repeated samples) of the ordinary least squares estimates of the regression coefficients follow a normal distribution. (by the **Central Limit Theorem**) That is -> All variables should be **multivariate normal** (i.e linear combination of r.v.s should be normally distributed). Checks for normality:

I. **Histogram**

II. QQ or **Quantile Quantile Plots** (A Q-Q plot is a scatterplot created by plotting two sets of quantiles against one another. If both sets of quantiles came from the same distribution, we should see the points forming a line that's roughly straight.)

III. **Kolmogorov-Smirnov test** (Goodness to fit test): Test for differences in shape for two sample distribution OR checks if a variable follows a given distribution in a population

When the data is not normally distributed a **non-linear transformation** (e.g., log-transformation) might fix this issue.

Q) What happens if the residuals are not normally distributed?
Ans) As statisticians we want to understand some of the properties of this method, answers to questions such as: **are the least squares estimators optimal in some sense**? Or can we do better with some alternative estimators? Then, under the normal distribution of error terms, we can show that these estimators are, indeed, optimal, for instance they are "unbiased of minimum variance", or maximum likelihood. No such thing can be proved without the normal assumption.

Also, if we want to construct (and analyze properties of) **confidence intervals or hypothesis tests, then we use the normal assumption**. But, we could instead construct confidence intervals by some other means, such as bootstrapping. Then, we do not use the normal assumption, but, alas, without that, it could be we should use some other estimators than the least squares ones, maybe some robust estimators?

Note: This is a weak assumption - violating normality is not seen as a big problem, specially for large no of observations.

c) There is little or **NO multicollinearity** in the data. Can be tested using:

I. **Correlation Matrix**- Pearson Correlation
II. **Tolerance** – the tolerance measures the influence of one independent variable on all other independent variables; the tolerance is calculated with an initial linear regression analysis. Tolerance/VIFs are calculated by taking a predictor, and regressing it against every other predictor in the model. This gives you the R-squared values, which can then be plugged into the Tolerance/VIF formula.
Tolerance is defined as $T = 1 – Ri^2$ for these first step regression analysis. With $T < 0.1$ there might be multicollinearity in the data and with $T < 0.01$ there certainly is.

III. **Variance Inflation Factor (VIF)** – the variance inflation factor of the linear regression is defined as VIF = 1/T. With VIF > 5 there is an indication that multicollinearity may be present; with VIF > 10 there is certainly multicollinearity among the variables.
This is used when we want to check correlation between more than 2 variables. Correlation matrices can give correlation between only 2 variables.

Q) How to treat multicollinearity?
If multicollinearity is found in the data, centering the data, that is deducting the mean score might help to solve the problem.
Dropping one of the highly correlated variables / combining them also helps.

Q) Why is removing highly correlated features important?
The interpretation of a regression coefficient is that it represents the mean change in the target for each unit change in a feature when you hold all of the other features constant. However, when features are correlated, changes in one feature in turn shifts another feature/features. The stronger the correlation, the more difficult it is to change one feature without changing another. It becomes difficult for the model to estimate the relationship between each feature and the target independently because the features tend to change in unison.

d) Independent Error Terms: There is little or no **autocorrelation** in the error terms. Autocorrelation occurs when the residuals are not independent from each other. **The presence of correlation in error terms drastically reduces a model's accuracy**. This usually occurs in time series models where the next instant is dependent on the previous instant. In other words when the value of y(x+1) is not independent from the value of y(x).

Q) What is the issue if error terms are not independent?
Ans) Standard errors cannot be relied on.

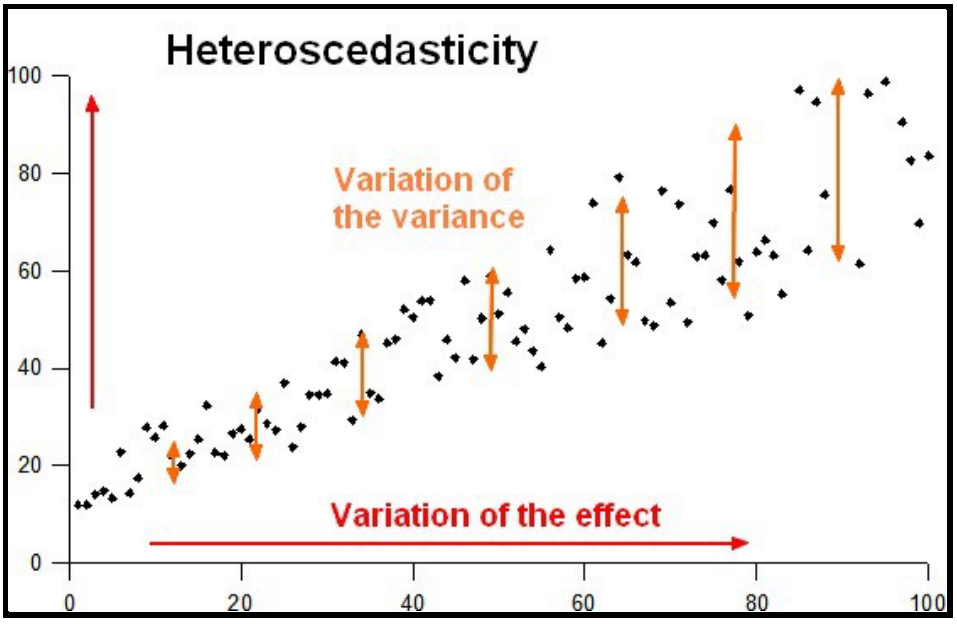
While a **scatterplot** allows you to check for autocorrelations, you can test the linear regression model for autocorrelation with the **Durbin-Watson test**. The null hypothesis of the test is that there is no serial correlation. The Durbin-Watson test statistics is defined as:

$$\sum_{t=2}^{T}((e_t - e_{t-1})^2) / \sum_{t=1}^{T} e_t^2$$

The test statistic is approximately equal to 2*(1-r) where r is the sample autocorrelation of the residuals. Thus, for r == 0, indicating no serial correlation, the test statistic equals 2. This statistic will always be **between 0 and 4**. The closer to 0 the statistic, the more evidence for positive serial correlation. The closer to 4, the more evidence for negative serial correlation.

e) Homoscedasticity (meaning residuals have "**same variance**"):
Homoscedasticity describes a situation in which the error term (that is, the "noise" or random disturbance in the relationship between the independent variables and the dependent variable) is the same across all values of the independent variables.

Q) How to check?
Ans) A scatter plot of residual values (Y-Axis) vs predicted values (X-Axis) is a good way to check for homoscedasticity.There should be no clear pattern (constant variance) in the distribution and if there is a specific pattern, the data is heteroscedastic.



Example: A simple **bivariate** example can help to illustrate heteroscedasticity: Imagine we have data on **family income and spending on luxury items.** Using bivariate

regression, we use family income to predict luxury spending. As expected, there is a strong, positive association between income and spending. Upon examining the residuals we detect a problem – the residuals are very small for low values of family income (almost all families with low incomes don't spend much on luxury items) while there is great variation in the size of the residuals for wealthier families (some families spend a great deal on luxury items while some are more moderate in their luxury spending). This situation represents **heteroscedasticity because the size of the error varies across values of the independent variable**. Examining a scatterplot of the residuals against the predicted values of the dependent variable would show a classic cone-shaped pattern of heteroscedasticity.

Q) Why is the assumption made?
Ans) The problem that heteroscedasticity presents for regression models is simple. Recall that ordinary least-squares (OLS) regression seeks to minimize residuals and in turn produce the smallest possible standard errors. **By definition, OLS regression gives equal weight to all observations, but when heteroscedasticity is present, the cases with larger disturbances have more "pull" than other observations**. In this case, weighted least squares regression would be more appropriate, as it down-weights those observations with larger disturbances.

Q) How to tackle it?
Ans) Another approach for dealing with heteroscedasticity is to transform the dependent variable using one of the **variance stabilizing transformations**. A logarithmic transformation can be applied to highly skewed variables, while count variables can be transformed using a square root transformation. Overall however, the violation of the homoscedasticity assumption must be quite severe in order to present a major problem given the robust nature of OLS regression.

f) Exogeneity (no omitted variable bias): If there is a variable that is emitted from the model, but affects both X and Y, then it could cause omitted variable bias.

Q) What is the issue?
Ans) Though we could still use this model for predictive purposes, but not for inferring causation purposes.

Q) How to detect?
Ans) Using intuition

Recommendations to improve the performance of the linear regression model:

1. Performing **Mean Normalization** to the predictors helps **gradient descent** to converge faster.

2. If the distribution of predictors are skewed, the coefficients estimated will also be skewed. Hence, it is a recommended practice to check for **skewness** in numerical predictors and making it symmetric to obtain improved results.

3. **Outlier** removal: An outlier is a point for which the actual response value is far from the predicted value by the regression model. Regression line does not have much effect on non removal of outliers, but affects the interpretation of the model as RSE and p-value corresponding to the model changes drastically.
   Residual plots are used to detect the outliers. Most of the statistical tools use the residuals to detect outliers.

4. High **leverage** observation removal: An outlier has an unusual value of response, whereas a high leverage observation has an **unusual value of the independent variable**. This has a substantial effect on regression fit, which in turn affects the coefficients.

5. **Influential** observation removal: Influential observation is one which has the **combined effect of both an outlier and a high leverage point**. Cook's Distance is calculated to determine the influential observations.

Model Diagnostics:

1. Test for overall model (R^2 and adjusted-R^2).
2. Test for statistical significance of overall model (F-statistic and it's p-value).
3. Test for statistical significance of coefficient of each predictor (t-statistic and it's corresponding p-value).
4. Test for normality(QQ-plot) and homoscedasticity(residual plot) of residuals and auto correlation(Durbin-watson statistic).
5. Test for multicollinearity.
6. Test for outliers, high leverage points and influential observations.

Model Selection:

Subset selection: Out of k predictors, only a subset of predictors might contribute to the Linear regression model and rest all could be noise to the data which might actually underestimate or overestimate the errors.

To select the best subset of variables, test error needs to be estimated either by directly estimating the test error through cross validation or by indirectly making an adjustment to the training error using the following metrics as discussed below.

Metrics to select the best subset:

- **Mallow's Cp:** Estimate of the size of bias introduced into the predicted response. Lower the Cp, better the model.
- **Akaike information criterion (AIC):** Amount of information lost due to the predictions on the response variable. AICs' main goal is to build model that effectively predicts the response variable.
- **Bayesian information criterion (BIC):** BIC is similar to AIC, but BIC penalizes the noise predictors. BICs' main goal is to extract the features that are actually influencing the response variable.
- **Adjusted-R^2:** Proportion of the response variable explained by the independent variables.

1. **Best subset selection:** Consider all possible combinations of predictors and estimate the test error indirectly by using the above mentioned metrics (Cp, AIC, BIC, R^2) or directly through cross validation and choose the best subset which has the least error. This method works when there are **fewer variables** as selection gets cumbersome if number of predictors are high (Exhaustive search).
2. **Forward stepwise selection:** Start with one variable with least test error and keep adding one predictor at a time with out removing any variable after a variable is added until all the variables are exhausted or the test error does not decrease.
3. **Backward stepwise selection:** Start with all the variables and remove one predictor at a time which results in least test error without adding any variable after removal until the test error does not decrease.
4. **Hybrid stepwise selection:** Combination of forward and backward stepwise selections where in at a particular step a variable can be dropped or added to improve the performance on the test data.

3. What is the maximum likelihood method?

Probability concepts explained: Maximum likelihood estimation | by Jonny Brooks-Bartlett | Towards Data Science

Ans: Maximum likelihood estimation is a method that **determines values for the parameters of a model.** The parameter values are found such that they maximise the likelihood that the process described by the model produced the data that were actually observed.

Likelihood function = L(Parameters(theta) | Given the data we have observed(X1,X2,...Xn) = Joint PMF of X1,X2…, Theta
= f(X1,X2..,Xn | theta) = f(X1 | theta) * f(X2 | theta) * …….

Max. Likelihood Estimator (MLE) is the value of theta that maximizes L

- We first have to decide which model we think best describes the process of generating the data. This part is very important. At the very least, we should have a good idea about which model to use. For these data we'll assume that the data generation process can be adequately described by a Gaussian (normal) distribution. **We want to know which curve was most likely responsible for creating the data points that we observed?** (See figure below). Maximum likelihood estimation is a method that will find the values of μ and σ that result in the curve that best fits the data.

- After deciding which curve was responsible for creating the data points that we observed, we want to calculate the parameter values.
  Again we'll demonstrate this with an example. Suppose we have three data points this time and we assume that they have been generated from a process that is adequately described by a Gaussian distribution. These points are 9, 9.5 and 11. **How do we calculate the maximum likelihood estimates of the parameter values of the Gaussian distribution μ and σ?**

What we want to calculate is the total probability of observing all of the data, i.e. the joint probability distribution of all observed data points. To do this we would need to calculate some conditional probabilities, which can get very difficult. So it is here that we'll make our first **assumption**. The assumption is that e**ach data point is generated independently of the others**. (**iids**: independent and identically distributed)

$$P(9, 9.5, 11; \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(9-\mu)^2}{2\sigma^2}\right) \times \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(9.5-\mu)^2}{2\sigma^2}\right) \times \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(11-\mu)^2}{2\sigma^2}\right)$$

We just have to figure out the values of μ and σ that result in giving the maximum value of the above expression.

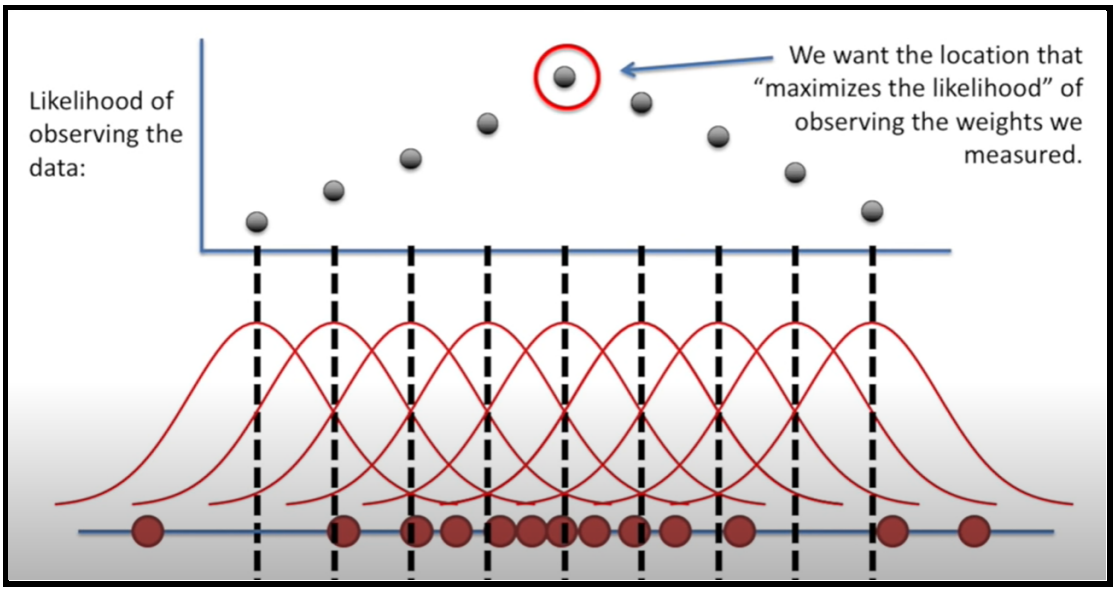


Figure: Finding the parameter Mu (mean) which maximizes the likelihood of observing the weight (or any other variable/feature) values

The above expression for the total probability is actually quite a pain to differentiate, so it is almost always simplified by taking the **natural logarithm** of the expression. This is absolutely fine because the natural logarithm is a **monotonically increasing** function. This means that if the value on the x-axis increases, the value on the y-axis also increases (see figure below). This is important because it ensures that the maximum value of the log of the probability occurs at the same point as the original probability function. Therefore we can work with the simpler **log-likelihood instead of the original likelihood**.

Q) So why maximum likelihood and not maximum probability?

$$L(\mu, \sigma; data) = P(data; \mu, \sigma)$$

Ans) These expressions are equal! So what does this mean? Let's first define P(data; μ, σ)? It means "the probability density of observing the data with model parameters μ and σ". It's worth noting that we can generalise this to any number of parameters and any distribution.
On the other hand L(μ, σ; data) means "the likelihood of the parameters μ and σ taking certain values given that we've observed a bunch of data."
The equation above says that the probability density of the data given the parameters is equal to the likelihood of the parameters given the data. But despite these two things being equal, the **likelihood and the probability density are fundamentally asking different questions** — one is asking about the data and the other is asking about the parameter values. This is why the method is called maximum likelihood and not maximum probability.

MLE for Linear Regression

What kind of distribution are we going to use in linear regression? With that question, we can talk about the main assumption in linear regression:
"The independent variable (y values) is assumed be in a normal distribution"

Since, Y = a + bx + error(e), where e ~ N(0, sigma2)
Y = constant + Normal Dist. = Normal Dist.
Find the mean, variance of Y -> E(Y) = E(a+bx+e) = a+bx+E(e) = a+bx
Var(Y) = Var(a+bx+e) = Var(e) = sigma2

Hence: $$y \sim N(\theta_1 x + \theta_0, \sigma^2)$$

$$f(y|x; \theta_0, \theta_1, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(y-(\theta_1 x+\theta_0))^2}{2\sigma^2}}$$

$$L_X(\theta_0, \theta_1, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \prod_{(x,y)\in X} e^{\frac{-(y-(\theta_1 x+\theta_0))^2}{2\sigma^2}}$$

$$l_X(\theta_0, \theta_1, \sigma^2) = \log[\frac{1}{\sqrt{2\pi\sigma^2}} \prod_{(x,y)\in X} e^{\frac{-(y-(\theta_1 x+\theta_0))^2}{2\sigma^2}}]$$

$$= \log(\frac{1}{\sqrt{2\pi\sigma^2}}) + \sum_{(x,y)\in X} \log(e^{\frac{-(y-(\theta_1 x+\theta_0))^2}{2\sigma^2}})$$

$$= \log(\frac{1}{\sqrt{2\pi\sigma^2}}) + \sum_{(x,y)\in X} \frac{-(y-(\theta_1 x+\theta_0))^2}{2\sigma^2}$$

$$= \log(1) - \log(\sqrt{2\pi\sigma^2}) - \frac{1}{2\sigma^2} \sum_{(x,y)\in X} [y - (\theta_1 x + \theta_0)]^2$$

$$= -\log(\sqrt{2\pi\sigma^2}) - \frac{1}{2\sigma^2} \sum_{(x,y)\in X} [y - (\theta_1 x + \theta_0)]^2$$

Log Likelihood: $$l_X(\theta_0, \theta_1, \sigma^2) = -\log(\sqrt{2\pi\sigma^2}) - \frac{1}{2\sigma^2} \sum (y - \hat{y})^2$$

Which is same as minimizing the sum of square error: $\sum (y - \hat{y})^2$

Q) When is least squares minimisation the same as maximum likelihood estimation?

Ans) Least squares minimisation is another common method for estimating parameter values for a model in machine learning. It turns out that when the model is assumed to be **Gaussian** as in the examples above, the **MLE estimates are equivalent to the least squares method**.

Intuitively we can interpret the connection between the two methods by understanding their objectives. For least squares parameter estimation we want to find the line that minimises the total squared distance between the data points and the regression line

(see the figure below). In maximum likelihood estimation we want to maximise the total probability of the data. When a Gaussian distribution is assumed, the maximum probability is found when the data points get closer to the mean value. Since the Gaussian distribution is symmetric, this is equivalent to minimising the distance between the data points and the mean value.

4. LASSO & Ridge Regression

https://www.youtube.com/watch?v=Q81RR3yKn30&list=PLblh5JKOoLUICTaGLRoHQDuF_7q2GfuJF&index=18

L2 / Ridge: Penalize the steeper slope by adding a term = lambda * (slope)^2
This will minimize slope/coefficient values -> very very close to zero

L1 / LASSO (Least Absolute Shrinkage and Selection Operator):
Term = lambda * abs(slope)
This will help in feature selection -> some slopes will become zero

Elastic-Net is a linear regression model trained with **both l1 and l2** -norm regularization of the coefficients.

Q) Difference between outlier and anomaly?
Outlier = **legitimate** data point that's far away from the mean or median in a distribution.
Anomaly = **illegitimate** data point that's generated by a different process than whatever generated the rest of the data.

5. Logistic Regression:

Logistic Regression — Detailed Overview | by Saishruthi Swaminathan | Towards Data Science

Logistic Regression for Machine Learning (machinelearningmastery.com)
4.2 Logistic Regression | Interpretable Machine Learning (christophm.github.io)

Interview-Prepartion-Data-Science/Interview Preparation- Day 5-Logistic Regression.ipynb at master · krishnaik06/Interview-Prepartion-Data-Science (github.com)

Logistic Function

Logistic regression is named for the function used at the core of the method, the logistic function. It's an **S-shaped curve** that can take any real-valued number and map it into a value between 0 and 1, but never exactly at those limits.

1 / (1 + e^-value)

What is Wrong with Linear Regression for Classification?

Technically it works and most linear model programs will spit out weights for you. But there are a few problems with this approach:

A linear model does not output probabilities, but it treats the classes as numbers (0 and 1) and fits the best hyperplane (for a single feature, it is a line) that minimizes the distances between the points and the hyperplane. So it simply interpolates between the points, and you cannot interpret it as probabilities.

A linear model also extrapolates and gives you **values below zero and above one**. This is a good sign that there might be a smarter approach to classification.

Since the predicted outcome is not a probability, but a linear interpolation between points, there is **no meaningful threshold at which you can distinguish one class from the other**.

Linear models do not extend to classification problems with **multiple classes**. You would have to start labeling the next class with 2, then 3, and so on. The classes might not have any meaningful order, but the linear model would force a weird structure on the relationship between the features and your class predictions. The higher the value of a feature with a positive weight, the more it contributes to the prediction of a class with a higher number, even if classes that happen to get a similar number are not closer than other classes.

The step from linear regression to logistic regression is kind of straightforward. In the linear regression model, we have modelled the relationship between outcome and features with a linear equation:

$$\hat{y}^{(i)} = \beta_0 + \beta_1 x_1^{(i)} + \ldots + \beta_p x_p^{(i)}$$

For classification, we prefer probabilities between 0 and 1, so we **wrap the right side of the equation into the logistic function**. This forces the output to assume only values between 0 and 1.

$$P(y^{(i)} = 1) = \frac{1}{1 + exp(-(\beta_0 + \beta_1 x_1^{(i)} + \ldots + \beta_p x_p^{(i)}))}$$

The **loss function** computes the error for a **single training example**; the **cost function** is the **average** of the loss functions of the entire training set.

Loss function: L(y',y) = - (y*log(y') + (1-y)*log(1-y'))
y* is the actual value, y' is the predicted value

Representation Used for Logistic Regression
Logistic regression uses an equation as the representation, very much like linear regression. Input values (x) are combined linearly using weights or coefficient values (referred to as the Greek capital letter Beta) to predict an output value (y). A key difference from linear regression is that the output value being modeled is a binary value (0 or 1) rather than a numeric value.

Logistic Regression Predicts Probabilities (Technical Interlude)
Logistic regression models the probability of the default class (e.g. the first class).

Logit or log(odds)
ln(p(X) / 1 – p(X)) = b0 + b1 * X

This is useful because we can see that the calculation of the output on the right is linear again (just like linear regression), and the input on the left is a log of the probability of the default class.

This ratio on the left is called the odds of the default class (it's historical that we use odds, for example, odds are used in horse racing rather than probabilities). Odds are calculated as a **ratio of the probability of the event divided by the probability of not the event**, e.g. 0.8/(1-0.8) which has the odds of 4. So we could instead write:

ln(odds) = b0 + b1 * X

Utility: Logit / log(odds) can be used to find out the coefficients/parameters of the model.

Gradient Descent

We want to predict w and b that minimize the cost function. Our **cost function is convex**. First we initialize w and b to 0,0 or initialize them to a random value in the convex function and then try to improve the values the reach minimum value. In Logistic regression people always use 0,0 instead of random.

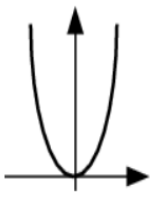
The gradient decent algorithm repeats: **w = w - alpha * dw** where alpha is the learning rate and dw is the derivative of w (Change to w) The derivative is also the slope of w.

w = w - alpha * d(J(w,b) / dw) (how much the function slopes in the w direction)
b = b - alpha * d(J(w,b) / db) (how much the function slopes in the d direction)

Convex Function:

Convex Function

DOWNLOAD
Wolfram Notebook



convex function

A convex function is a continuous function whose value at the midpoint of every interval in its domain does not exceed the arithmetic mean of its values at the ends of the interval.

More generally, a function $f(x)$ is convex on an interval $[a, b]$ if for any two points x_1 and x_2 in $[a, b]$ and any λ where $0 < \lambda < 1$,

$$f[\lambda x_1 + (1-\lambda)x_2] \le \lambda f(x_1) + (1-\lambda) f(x_2)$$

(Rudin 1976, p. 101; cf. Gradshteyn and Ryzhik 2000, p. 1132).

If $f(x)$ has a second derivative in $[a, b]$, then a necessary and sufficient condition for it to be convex on that interval is that the second derivative $f''(x) \ge 0$ for all x in $[a, b]$.

If the inequality above is strict for all x_1 and x_2, then $f(x)$ is called strictly convex.

Important Questions and Answers:

Q) Is it possible to apply a logistic regression algorithm on a 3-class Classification problem?
Ans) Yes, we can apply logistic regression on 3 classification problem, We can use **One v/s All method** for 3 class classification in logistic regression.

Q) Which of the following methods do we use to best fit the data in Logistic Regression?
Ans) Logistic regression uses maximum likelihood estimate for training a logistic regression.

Q) Do errors in Logistic regression have to be normally distributed?
A) Linear Regression **errors values** has to be normally distributed but in case of Logistic Regression it is not the case

Q) How many models are required for a n-class One-vs-All method?
Ans) We need to fit **n models** in n-class classification problem

Q) Why is accuracy not a good measure for classification problems?
Ans) Accuracy is not a good measure for classification problems because it gives **equal importance to both false positives and false negatives**. However, this may not be the case in most business problems. For example, in case of cancer prediction, declaring cancer as benign is more serious than wrongly informing the patient that he is suffering from cancer. Accuracy gives equal importance to both cases and cannot differentiate between them

Q) Why can't we use Mean Square Error (MSE) as a cost function for logistic regression?
Ans) In logistic regression, we use the sigmoid function and perform a non-linear transformation to obtain the probabilities. Squaring this non-linear transformation will lead to **non-convexity with local minimums**. Finding the global minimum in such cases using gradient descent is not possible. Due to this reason, MSE is not suitable for logistic regression. **Cross-entropy or log loss** is used as a cost function for logistic regression.

Q) What is the importance of a baseline in a classification problem?

Most classification problems deal with **imbalanced datasets**. Examples include telecom churn, employee attrition, cancer prediction, fraud detection, online advertisement targeting, and so on. In all these problems, the number of the positive classes will be very low when compared to the negative classes. In some cases, it is common to have positive classes that are less than 1% of the total sample. In such cases, an accuracy of 99% may sound very good but, in reality, it may not be. Here, the negatives are 99%, and hence, the baseline will remain the same. If the algorithms predict all the instances as negative, then also the accuracy will be 99%. In this case, all the positives will be predicted wrongly, which is very important for any business. Even though all the positives are predicted wrongly, an accuracy of 99% is achieved. So, the baseline is very important, and the **algorithm needs to be evaluated relative to the baseline.**

Q) How to choose a cutoff point in case of a logistic regression model?
The cutoff point depends on the **business objective**. Depending on the goals of your business, the cutoff point needs to be selected. For example, let's consider **loan defaults**. If the business objective is to reduce the loss, then the sensitivity (considering default = 1, not default = 0) needs to be high. If the aim is to increase profits, then it is an entirely different matter. It may not be the case that profits will increase by avoiding giving loans to all predicted default cases. But it may be the case that the business has to disburse loans to default cases that are slightly less risky to increase the profits. In such a case, a different cutoff point, which maximises profit, will be required. In most of the instances, businesses will operate around many constraints. The cutoff point that satisfies the business objective will not be the same with and without limitations. The cutoff point needs to be selected considering all these points. **As a thumb rule, choose a cutoff value that is equivalent to the proportion of positives in a dataset.**

Q) How does logistic regression handle categorical variables?
Ans) The inputs to a logistic regression model need to be numeric. The algorithm **cannot handle categorical variables directly.** So, they need to be converted into a format that is suitable for the algorithm to process. The various levels of a categorical variable will be assigned a unique numeric value known as the **dummy variable**. These dummy variables are handled by the logistic regression model as any other numeric value.

Q) Which algorithm is better at handling outliers logistic regression or SVM?
Ans) Logistic regression will find a linear boundary if it exists to accommodate the outliers. Logistic regression will shift the linear boundary in order to accommodate the outliers. **SVM is insensitive to individual samples**. There will not be a major shift in

the linear boundary to accommodate an outlier. SVM comes with inbuilt complexity controls, which take care of overfitting. This is not true in case of logistic regression.

Q) How will you deal with the multiclass classification problem using logistic regression?
Ans) The most famous method of dealing with multiclass classification using logistic regression is using the **one-vs-all approach**. Under this approach, a number of models are trained, which is equal to the number of classes. The models work in a specific way. For example, the first model classifies the datapoint depending on whether it belongs to class 1 or some other class; the second model classifies the datapoint into class 2 or some other class. This way, each data point can be checked over all the classes.

Q) Explain the use of ROC curves and the AUC of an ROC Curve.
Ans) An ROC (Receiver Operating Characteristic) curve illustrates the performance of a binary classification model. It is basically a **TPR versus FPR** (true positive rate versus false-positive rate) curve for **all the threshold values ranging from 0 to 1.** In a ROC curve, each point in the ROC space will be associated with a different confusion matrix. A diagonal line from the bottom-left to the top-right on the ROC graph represents random guessing. The Area Under the Curve (AUC) signifies how good the classifier model is. If the value for AUC is high (near 1), then the model is working satisfactorily, whereas if the value is low (around 0.5), then the model is not working properly and just guessing randomly.

Stepwise Regression

Stepwise Regression - Statistics How To

Stepwise regression is the step-by-step iterative construction of a regression model that involves the selection of independent variables to be used in a final model. It involves adding or removing (usually via a series of F-tests or T-tests. The variables to be added or removed are chosen based on the test statistics of the estimated coefficients) potential explanatory variables in succession and testing for statistical significance after each iteration.

6. Probabilistic Model Selection with AIC, BIC, and MDL

Probabilistic Model Selection with AIC, BIC, and MDL (machinelearningmastery.com)

RPubs - ML - Probabilistic Model Selection with AIC, BIC

The Challenge of Model Selection

Model selection is the process of fitting multiple models on a given dataset and choosing one over all others.

There are many common approaches that may be used for model selection. For example, in the case of supervised learning, the three most common approaches are:

1. Train, Validation, and Test datasets.
2. Resampling Methods.
3. Probabilistic Statistics.

The simplest reliable method of model selection involves fitting candidate models on a training set, tuning them on the validation dataset, and selecting a model that performs the best on the test dataset according to a chosen metric, such as accuracy or error. A problem with this approach is that it requires a lot of data.

Resampling techniques attempt to achieve the same as the train/val/test approach to model selection, although using a small dataset. An example is k-fold cross-validation where a training set is split into many train/test pairs and a model is fit and evaluated on each. This is repeated for each model and a model is selected with the best average score across the k-folds. A problem with this and the prior approach is that only model performance is assessed, regardless of model complexity.

A third approach to model selection attempts to combine the complexity of the model with the performance of the model into a score, then select the model that minimizes or maximizes the score.

Models are scored both on their performance on the training dataset and based on the complexity of the model.

a. Model **Performance**. How well a candidate model has performed on the training dataset.
b. Model **Complexity**. How complicated the trained candidate model is after training.

A benefit of probabilistic model selection methods is that a test dataset is not required, meaning that all of the data can be used to fit the model, and the final model that will be used for prediction in the domain can be scored directly.

The limitations:
- the same general statistic cannot be calculated across a range of different types of models. i.e. the metric must be carefully derived for each model.
- do not take the uncertainty of the model into account, and in practice they tend to favour overly simple models.

Akaike Information Criterion

$$AIC = -\frac{2}{N} \times LL + 2 \times \frac{k}{N}$$

where
N: the number of examples in the training dataset,
LL: the log-likelihood of the model on the training dataset,
k: the number of parameters in the model.

$\Longrightarrow$ To use AIC for model selection, we simply choose the model giving smallest AIC over the set of models considered.

$\Longrightarrow$ The penalty for AIC is less than for BIC. This causes AIC to pick more complex models.

Bayesian Information Criterion

$$BIC = -2 \times LL + log(N) \times k$$

where
log(): the natural logarithm N: the number of examples in the training dataset,
LL: the log-likelihood of the model on the training dataset,
k: the number of parameters in the model.

$\Longrightarrow$ score as defined above is minimized, e.g. the model with the lowest BIC is selected

$\Longrightarrow$ the BIC penalizes the model more for its complexity, meaning that more complex models will have a worse (larger) score and will, in turn, be less likely to be selected $\Longrightarrow$ given a family of models, including the true model, the probability that BIC will select the correct model approaches one as the sample size N $\rightarrow$ infinity $\longrightarrow$ that for smaller, less representative training datasets, it is more likely to choose models that are too simple.

Minimum Description Length

$$MDL = L(h) + L(D|h)$$

Where
h: the model,
D: the predictions made by the model,
L(h): the number of bits required to represent the model, and L(D | h): the number of bits required to represent the predictions from the model on the training dataset.

In words, The Minimum Description Length (MDL) principle recommends choosing the hypothesis that minimizes the sum of these two description lengths.

The MDL calculation is very similar to BIC and can be shown to be equivalent in some situations.

Q) What are the advantages and disadvantages of solely using correlation values of features with target value for feature selection ?

7. Statistics:

Difference between Data Science and Statistics?

DS - Data is given, you need to ask the right questions
Stats - What data should we collect to answer this question?

Types of Stats

1. Descriptive
2. Predictive
3. Prescriptive

Causality vs Correlation

Sampling Bias-

Response Bias
Convenience bias
Voluntary response bias

Random Sampling-

Simple Random Sample
Stratification
Clustering

Types of Studies-

1. Sampling Study: Estimate a parameter of the population using a sample
2. Observational study: Correlations
3. Experiment: Establish Causality -> Control and treatment group
   Matched pair experiment design

Sampling Distributions:

If np >= 10 AND n(1-p) >= 10 then the sampling distribution is approximately normal

Mean of sampling distribution = Mean of population
Std Deviation of sampling distribution = Std Deviation of Population / (root n)

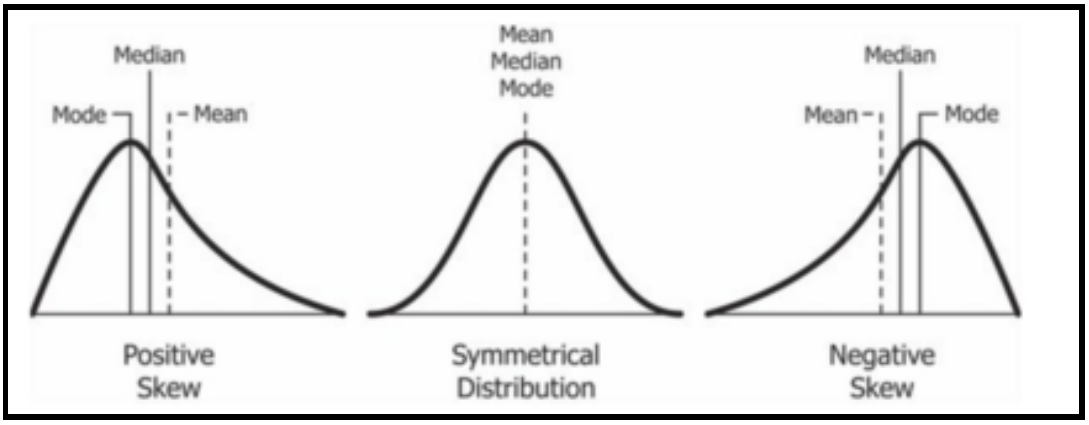
8. Central Limit Theorem:

In the study of probability theory, the central limit theorem (CLT) states that the **distribution of sample approximates a normal distribution** (also known as a "bell curve") as the **sample size becomes larger**, assuming that all samples are identical in size, and regardless of the population distribution shape.

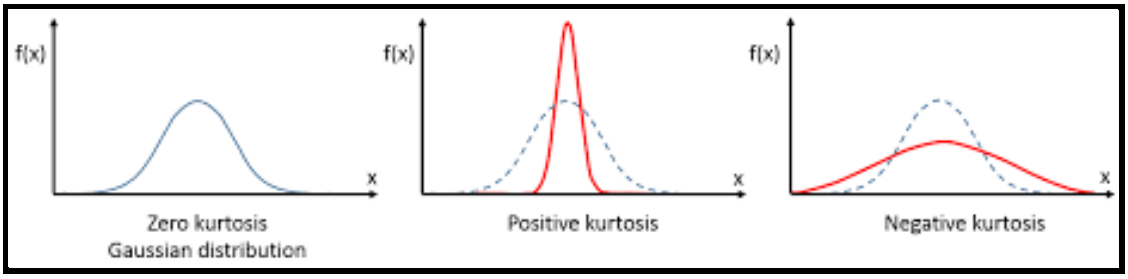
Sample sizes equal to or **greater than 30** are considered sufficient for the CLT to hold.

A key aspect of CLT is that the average of the sample means and standard deviations will equal the population mean and standard deviation.

Right/Positive **Skew**: Long Right Tail
Left Skew: Long left tail



Positive **Kurtosis** - Fatter tail, more pointy peak
Negative Kurtosis - Thinner Tail, less pointy peak



Standard Error of mean: The standard deviation (SD) measures the amount of variability, or dispersion, from the individual data values to the mean, while the standard error of the mean (SEM) measures how far the sample mean (average) of the data is likely to be from the true population mean. The SEM is always smaller than the SD.

The Standard Error ("Std Err" or "SE"), is an indication of the reliability of the mean. A small SE is an indication that the sample mean is a more accurate reflection of the actual population mean. A larger sample size will normally result in a smaller SE (while SD is not directly affected by sample size).
SD tells us about the shape of our distribution, how close the individual data values are from the mean value. **SE tells us how close our sample mean is to the true mean of the overall population**. Together, they help to provide a more complete picture than the mean alone can tell us.

Calculating Standard Deviation

$$\text{standard deviation } \sigma = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1}}$$

$$\text{variance} = \sigma^2$$

$$\text{standard error } (\sigma_{\bar{x}}) = \frac{\sigma}{\sqrt{n}}$$

where:

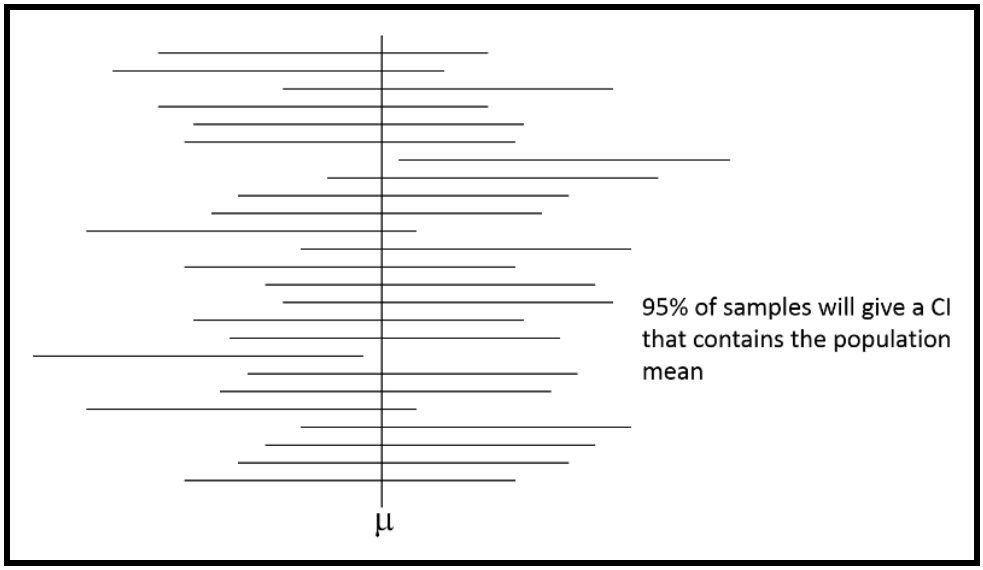
$\bar{x}$ = the sample's mean

n = the sample size

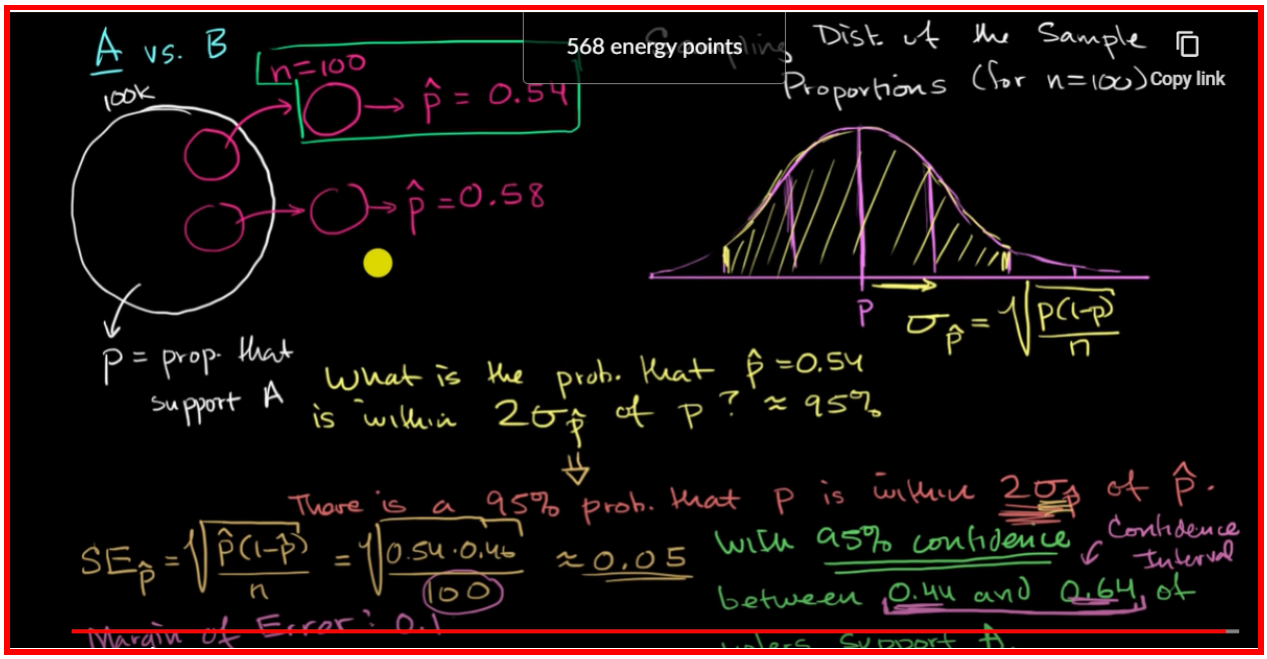
9. Confidence Interval:

https://www.khanacademy.org/math/statistics-probability/

Confidence intervals give us a **range of plausible values for some unknown parameter based on results from a sample**.
Strictly speaking a 95 % confidence interval means that if we were to take 100 different samples and compute a 95 % confidence interval for each sample, then approximately 95 of the 100 confidence intervals will contain the true mean (parameter) value (μ)



In practice, however, we select one random sample and generate one confidence interval, which may or may not contain the true mean. The observed interval may over- or underestimate μ. Consequently, the 95% CI is the likely range of the true, unknown parameter.



The confidence level refers to the long-term success rate of the method, that is, how often this type of interval will capture the parameter of interest.

When we want to carry out inferences on one proportion (build a confidence interval or do a significance test), the accuracy of our methods depend on a **few conditions**. Before doing the actual computations of the interval or test, it's important to check whether or not these conditions have been met, otherwise the calculations and conclusions that follow aren't actually valid. The conditions we need for inference on one proportion are:

Random: The data needs to come from a random sample or randomized experiment.

Normal: The **sampling distribution of p hat,** on top needs to be approximately normal — needs at least 10 expected successes and 10 expected failures. I.e. np^ >= 10 and n(1-p^) >= 10
OR
The original distribution needs to be normal
OR
n>30 (then acc to central limit theorem, the sampling distribution would be normal)
OR
The distribution is roughly Symmetric

Independent: Individual observations need to be independent (this condition is met when sampling with replacement). If sampling without replacement, our sample size shouldn't be more than **10%**, percent of the population.

To use the formula for standard deviation of p^, we need individual observations to be independent. When we are sampling without replacement, individual observations aren't technically independent since removing each item changes the population.
But the 10%, percent condition says that if we sample 10%, percent or less of the population, we can treat individual observations as independent since removing each observation doesn't significantly change the population as we sample. For instance, if our sample size is n=150, there should be at least N=1500 members in the population.
This allows us to use the formula for standard deviation of p^.

This allows us to use the formula for standard deviation of $\hat{p}$:

$$\sigma_{\hat{p}} = \sqrt{\frac{p(1-p)}{n}}$$

In a significance test, we use the sample size n and the hypothesized value of p.

If we are building a confidence interval for p, we don't actually know what p is, so we substitute $\hat{p}$ as an estimate for p. When we do this, we call it the **standard error** of $\hat{p}$ to distinguish it from the standard deviation.

So our formula for standard error of $\hat{p}$ is

$$\sigma_{\hat{p}} \approx \sqrt{\frac{\hat{p}(1-\hat{p})}{n}}$$

So the confidence interval is: (statistic)^ +/- (z*) times (Standard Deviation of population distribution)

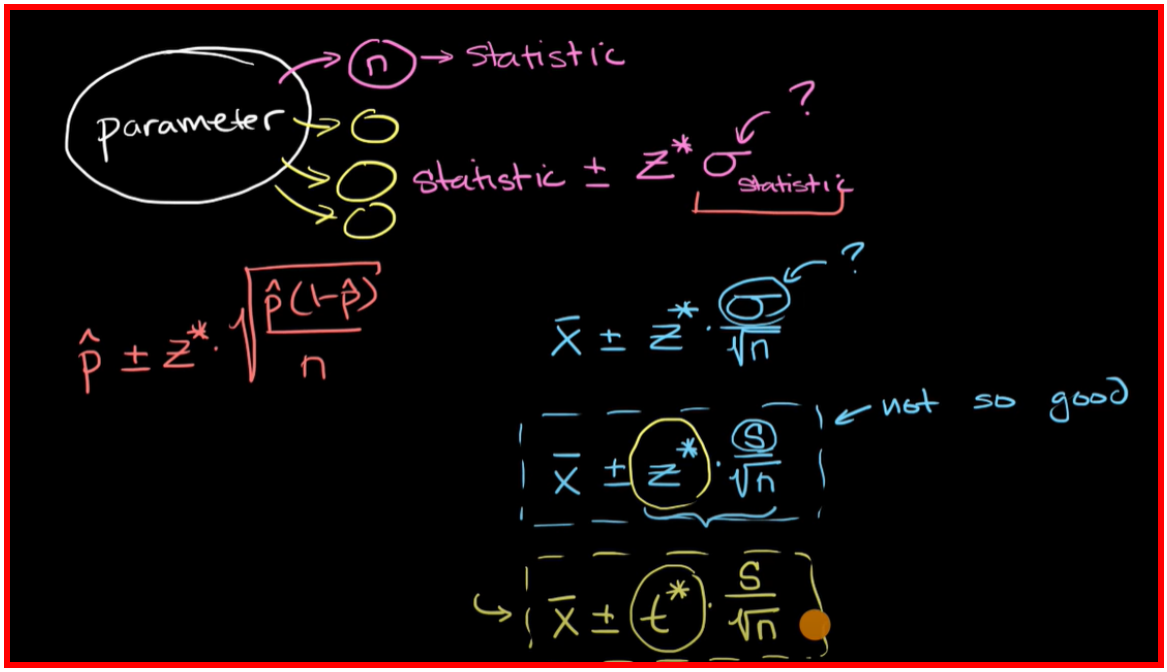
10. T-statistics:

In statistics, the t-statistic is the **ratio** of the departure of the **estimated** value of a parameter from its **hypothesized** value (usually the value hypothesized in null hypothesis) to its **standard error**. It is used in hypothesis testing via Student's t-test. The T-statistic is used in a T test to determine if you should support or reject the null hypothesis.

Let $\hat{\beta}$ be an estimator of parameter β in some statistical model. Then a *t*-statistic for this parameter is any quantity of the form

$$t_{\hat{\beta}} = \frac{\hat{\beta} - \beta_0}{\mathrm{s.e.}(\hat{\beta})}$$

It is **very similar to the Z-score** but with the difference that **T-statistic** is used when the **sample size is small (<30)** and the **population standard deviation is unknown**. For small sample size (<30), the standard deviation of the population cannot be estimated using the std. deviation of the sample. For example, the T-statistic is used in estimating the population mean from a sampling distribution of sample means if the population standard deviation is unknown. It is also used along with p-value when running hypothesis tests where the p-value tells us what the odds are of the results to have happened.



When sample size or n is very small (<30), we assume that the sampling distribution is a **t-distribution** (similar to normal dist. with **fatter tails**), not a normal distribution.
For sample size >30, we assume a normal distribution (according to the central limit theorem) and hence we can use the z-statistic.

For finding the t value from t-table we need the **degrees of freedom** which is equal to **(n-1)**

Q) When and why can't z-stat be used?

Ans) If the population std deviation (sigma) is known, we can find the std deviation of the sampling distribution (which is = Std. Error = Population Std. Deviation / (root(n)).
But in most cases, population std deviation is unknown, and hence we cannot estimate the std. deviation of sampling distribution (also called std. error) unless n>30 (then acc to CLT, we can use z-stat as the dist. will be normal).
In this case, we use the std. deviation of just the sample (denoted by "s") (just the sample, not sampling dist.) along with the t-stat (not z-stat) because t-stat gives a better probability than z-stat in this case.

Conclusion: If n>30, use z-stat (dist. becomes normal acc to CLT and hence population std deviation can be estimated using sample std deviation)
Else if n<30, the dist is not normal but a t-dist (fat tails), and hence use the t-stat not z-stat.

11. Hypothesis Testing & P-Value:

Evaluates two mutually exclusive statements about population data using sample data.

Step 1: Define the null and alternate hypothesis
Claims about the population you care about (not sample statistics)
Null Hypothesis: "no difference" hypo - things are happening as expected
Alternate Hypothesis: Suspect something different than what is expected

Significance Testing:

Step 2: Set a Threshold = Significance level (denoted by alpha) (example alpha = 0.05)
Step 3: Take a sample -> Calculate stats -> If we assume that the null hypothesis is true, what is the probability of getting a sample with the statistics that we get?
If that prob. is lower than our significance level then we reject the null hypo and say that we have evidence for the alternative. But if the prob. equal or greater to our significance level, we say that we CANNOT REJECT the null hypothesis, we aren't able to have evidence for the alternate.

Step 4: Calculate the p-value (or probability value)

The P value, or calculated probability, is the probability of finding the observed, or more extreme, results when the null hypothesis (H0) of a study question is true

P-value = P(X bar >= sample mean | H0)

We use p-values to make conclusions in significance testing. More specifically, we compare the p-value to a significance level alpha to make conclusions about our hypotheses.

P-Value Approach

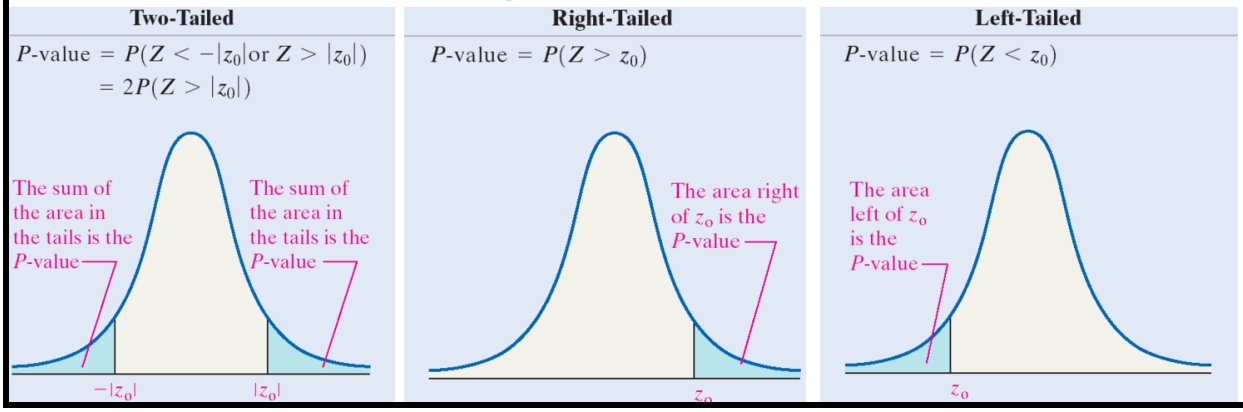
Assume that the null hypothesis is true.

The P-Value is the probability of observing a sample mean that is as or more extreme than the observed.

How to compute the P-Value for each type of test:

Step 1: Compute the test statistic $z_0 = \frac{\bar{x} - \mu_0}{\sigma / \sqrt{n}}$

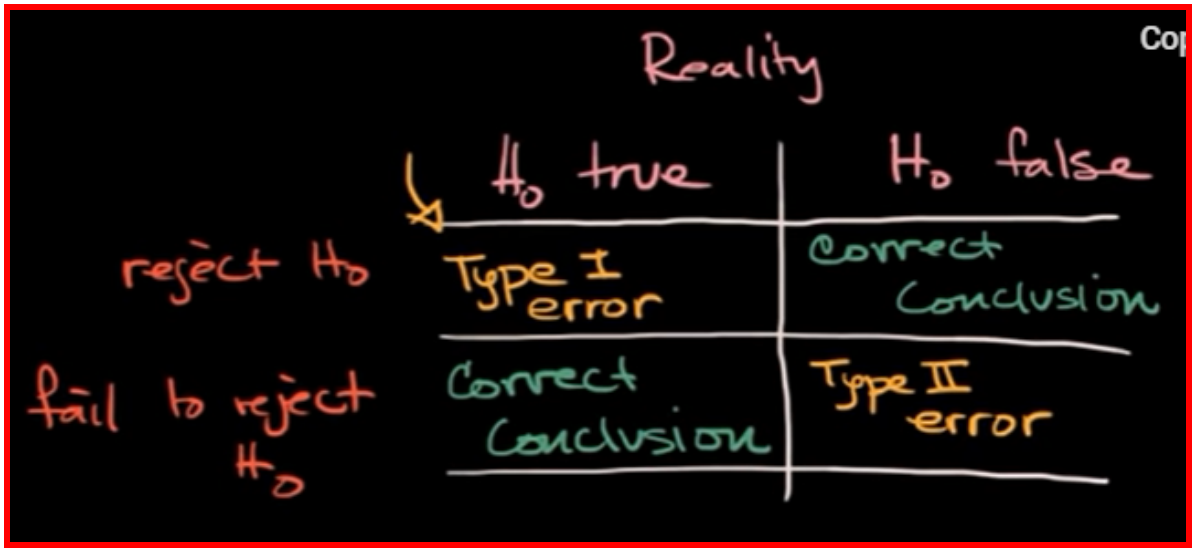
Two-tail | Right Tail | Left Tail



Step 5: Reject/ Fail to Null Hypothesis

If p-value < alpha -> Reject H0: The chance/prob. of getting such a statistic is very low if the null hypo. is assumed to be true and hence the null hypothesis cannot be true-> rejected.

If p-value >alpha -> Cannot Reject H0



Type 1 error: Rejecting the null hypothesis even though it's true. (False +ve)
Type 2 error: Failing to reject the null hypotheses even though it's false. (False -ve)

Alpha / Significance Level = P (Type 1 Error)

Power:

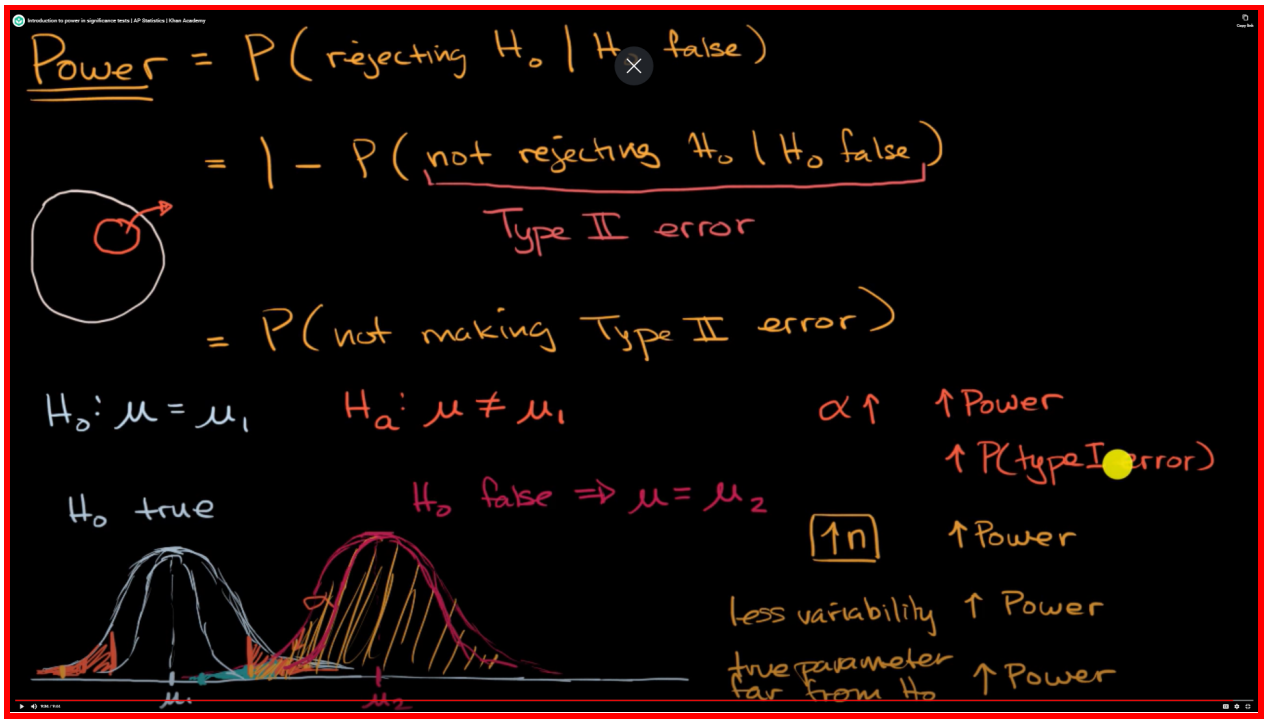
Power = P (rejecting the H0 | H0 = False)
= 1 - P (not rejecting the H0 | H0 = False)
= 1 - P (Type 2 Error)
= P (Not Making a Type 2 Error)

Increasing the alpha -> Increases the Power -> Decreases Type 2 Error
Increasing the alpha -> Increases Type 1 Error

Increasing sample size (n) -> Makes curve more normal / narrower -> Less Overlap -> Increases the power

Less Variability -> Increases Power

True parameter than what H0 says -> Increases Power



Two Tailed and One Tailed Tests

Graph in the sampling distribution -> both ends or only one end?
Depends on the alternate hypothesis: If Ha: X<10 or Ha>10 -> One tailed
If Ha: X not equal to 10 -> Two Tailed

Using t-stat and p-value to make inference on Slope of Linear Regression

H0 = Slope = 0 i.e the independent variable has no effect/relationship on/with dependent variable
Ha = Slope not equal 0 i.e has effect (significant independent variable)

Compare p-value corresponding to the t-stat to the significance level (alpha).
If p-value < alpha: Reject the H0 -> The independent variable is significant/important
If p-value > alpha: Fail to reject H0 -> Insignificant

Transforming Non-Linear Data

Using log, square, square root, etc. transformation to get linear models should be tried! Because we have developed so many tools etc around linear model, that it is always easier to do things with a linear model than an exponential, etc. model.

12. Chi-Squared Statistic, Test and Distribution

Chi-Square Statistic: How to Calculate It / Distribution - Statistics How To

A chi-square statistic is one way to show a **relationship between two categorical variables**. In statistics, there are two types of variables: numerical (countable) variables and non-numerical (categorical) variables. T**he chi-squared statistic is a single number that tells you how much difference exists between your observed counts and the counts you would expect if there were no relationship at all in the population.**

$$\chi_c^2 = \sum \frac{(O_i - E_i)^2}{E_i}$$

The subscript "c" is the degrees of freedom. "O" is your observed value and E is your expected value.

Q) What is a Chi Square Test?
Ans) There are **two types** of chi-square tests. Both use the chi-square statistic and distribution for different purposes:

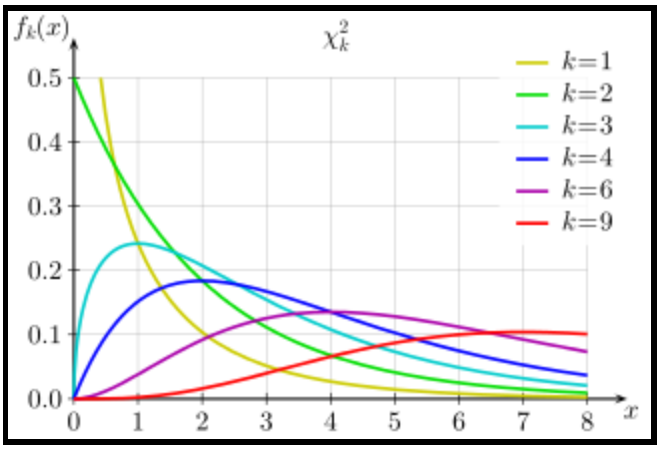
1. A chi-square **goodness of fit test** determines if **sample data matches a population**.
2. A chi-square test for **independence** compares two variables in a contingency table to see if they are related. In a more general sense, it tests to see whether distributions of categorical variables differ from each other.

A very **small** chi square test statistic means that your observed data fits your expected data extremely well. In other words, **there is a relationship**.
A very large chi square test statistic means that the data does not fit very well. In other words, there isn't a relationship.
if your observed and expected values were equal ("no difference -> high correlation") then chi-square would be zero — an event that is unlikely to happen in real life.

You could take your calculated chi-square value and compare it to a **critical value** from a chi-square table. If the chi-square value is more than the critical value, then there is a

significant difference. You could also use a **p-value**. First state the null hypothesis and the alternate hypothesis. Then generate a chi-square curve for your results along with a p-value (See: Calculate a chi-square p-value Excel). Small p-values (under 5%) usually indicate that a difference is significant (or “small enough”).

Chi Squared Distribution: Let’s say you have a random sample taken from a normal distribution. The chi square distribution is the distribution of the sum of these random samples squared . The degrees of freedom (k) are equal to the number of samples being summed.



The **degrees of freedom** in a chi square distribution is also its **mean**. Chi square distributions are always **right skewed**. However, the greater the degrees of freedom, the more the chi square distribution looks like a normal distribution.

13. ANOVA (Analysis of Variance)

ANOVA Test: Definition, Types, Examples - Statistics How To

Analysis of Variance (ANOVA) is a statistical method used to **test differences between two or more means**. It may seem odd that the technique is called "Analysis of Variance" rather than "Analysis of Means." As you will see, the name is appropriate because inferences about means are made by analyzing variance.

ANOVA tests the non-specific null hypothesis that all population means are equal.

Examples where ANOVA may be used:

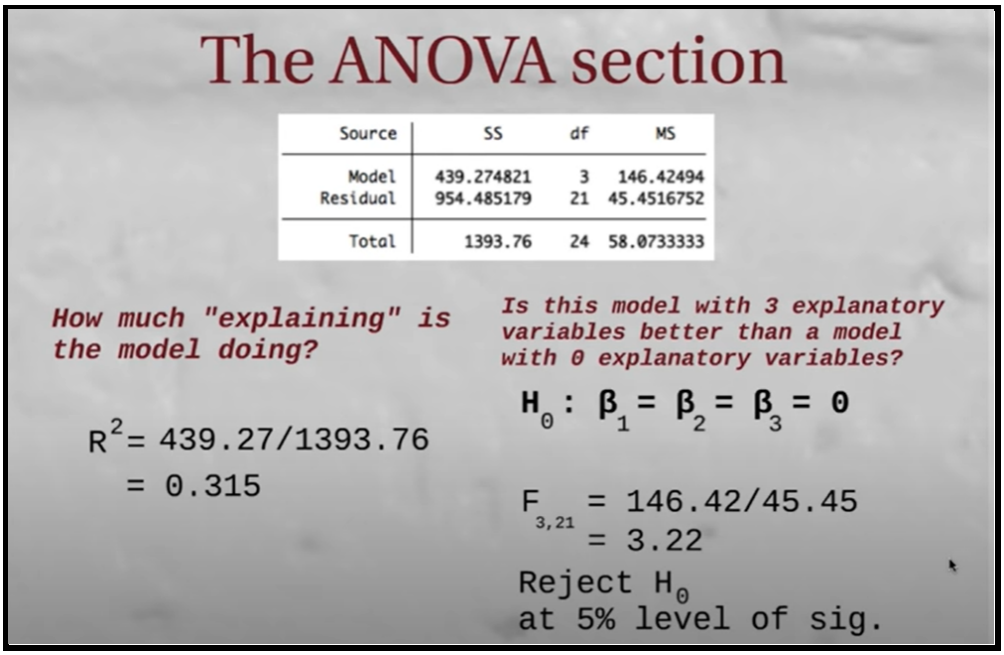
- A group of psychiatric patients are trying three different therapies: counseling, medication and biofeedback. You want to see if one therapy is better than the others.
- A manufacturer has two different processes to make light bulbs. They want to know if one process is better than the other.
- Students from different colleges take the same exam. You want to see if one college outperforms the other.

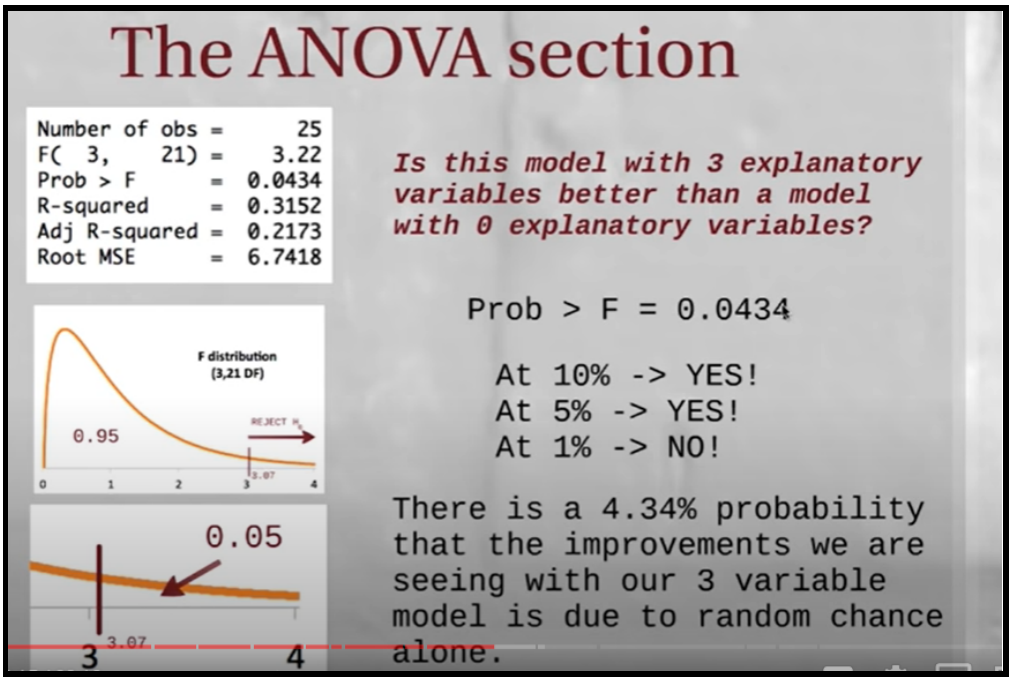
Q) What are "**Groups**" or "**Levels**"?
Groups or levels are **different groups within the same independent variable**. In the above example, your levels for "brand of cereal" might be Lucky Charms, Raisin Bran, Cornflakes — a total of three levels. Your levels for "Calories" might be: sweetened, unsweetened — a total of two levels.

Factor = Independent Variable
Groups / Levels = No of independent variables

An ANOVA conducted on a design in which there is only **one factor** (one independent variable) is called a **one-way** ANOVA
If an experiment has two factors, then the ANOVA is called a **two-way** ANOVA.

When different subjects are used for the levels of a factor, the factor is called a **between-subjects factor** or a between-subjects variable.
When the same subjects are used for the levels of a factor, the factor is called a **within-subjects factor** or a within-subjects variable.





Q) What Does "One-Way" or "Two-Way Mean?

One-way or **two-way** refers to the **number of independent variables** (IVs) in your Analysis of Variance test.

- One-way has one independent variable (with 2 levels). For example: brand of cereal,
- Two-way has two independent variables (it can have multiple levels). For example: brand of cereal, calories.

Types of Tests: There are two main types: one-way and two-way. Two-way tests can be with or without replication.

- One-way ANOVA between groups: used when you want to test two groups to see if there's a difference between them.
- Two way ANOVA without replication: used when you have one group and you're double-testing that same group. For example, you're testing one set of individuals before and after they take a medication to see if it works or not.
- Two way ANOVA with replication: Two groups, and the members of those groups are doing more than one thing. For example, two groups of patients from different hospitals trying two different therapies.

One Way ANOVA

A one way ANOVA is used to compare two means from two independent (unrelated) groups using the **F-distribution**. The null hypothesis for the test is that the two means are equal. Therefore, a significant result means that the two means are unequal.

Two Way ANOVA

A Two Way ANOVA is an extension of the One Way ANOVA. With a One Way, you have one independent variable affecting a dependent variable. With a Two Way ANOVA, there are two independents. **Use a two way ANOVA when you have one measurement variable (i.e. a quantitative variable) and two nominal variables**. In other words, if your experiment has a quantitative outcome and you have two categorical explanatory variables, a two way ANOVA is appropriate.

For example, you might want to find out if there is an interaction between income and gender for anxiety level at job interviews. The anxiety level is the outcome, or the variable that can be measured. Gender and Income are the two categorical variables. These categorical variables are also the independent variables, which are called factors in a Two Way ANOVA.

The factors can be split into levels. In the above example, income level could be split into three levels: low, middle and high income. Gender could be split into three levels:

male, female, and transgender. Treatment groups are all possible combinations of the factors. In this example there would be 3 x 3 = 9 treatment groups.

ANOVA vs. T Test

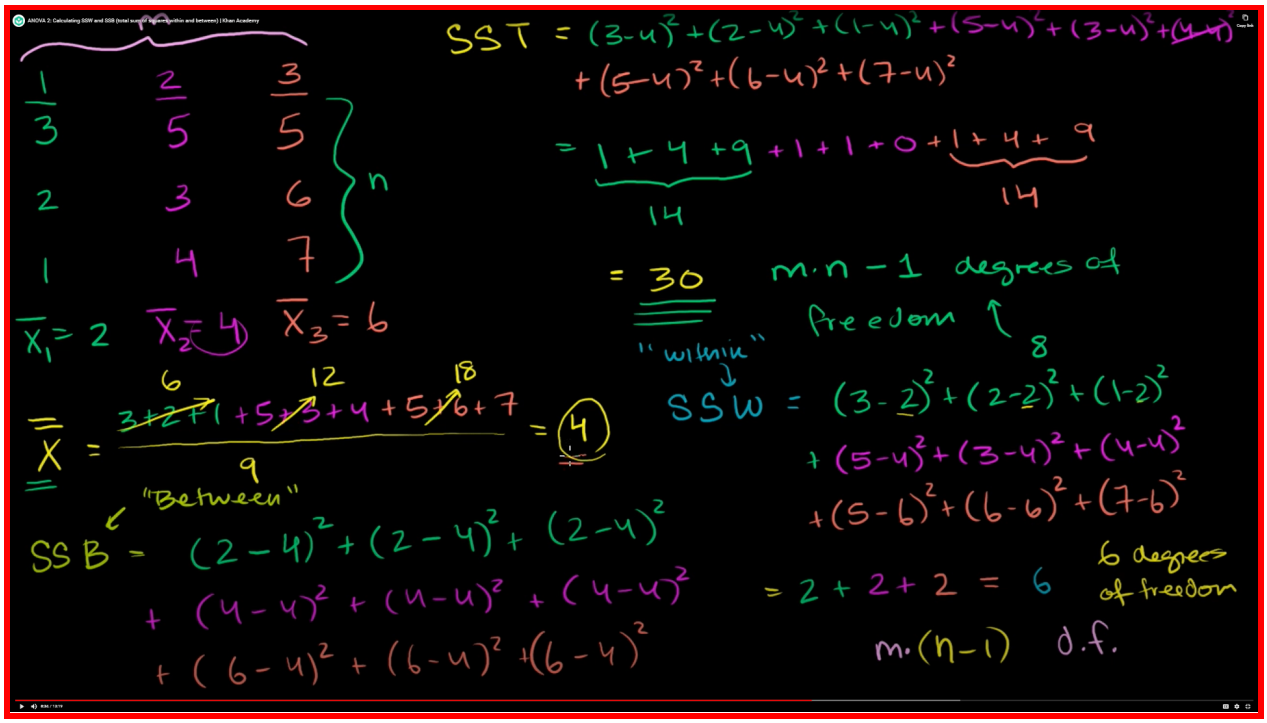
A Student's t-test will tell you if there is a significant variation between groups. A t-test compares means, while the ANOVA compares variances between populations.

You could technically perform a series of t-tests on your data. However, as the groups grow in number, you may end up with a lot of pair comparisons that you need to run. ANOVA will give you a single number (the f-statistic) and one p-value to help you support or reject the null hypothesis.

Q) Why not compare groups with multiple t-tests?

Ans) Every time you conduct a t-test there is a chance that you will make a Type I error. This error is usually 5%. By running two t-tests on the same data you will have increased your chance of "making a mistake" to 10%. The formula for determining the new error rate for multiple t-tests is not as simple as multiplying 5% by the number of tests. However, if you are only making a few multiple comparisons, the results are very similar if you do. As such, three t-tests would be 15% (actually, 14.3%) and so on. These are unacceptable errors. An ANOVA controls for these errors so that the Type I error remains at 5% and you can be more confident that any statistically significant result you find is not just running lots of tests.

Calculate SST, SSB, SSW and F-Statistic



SSW (Sum of Squares “Within” groups) ; dof = m.(n-1)
SSB (Sum of Squares “Between” groups) ; dof = m - 1
SST (Sum of Squares Total) = SSW + SSB ; dof = m.n - 1

F-stat = [SSB / (m -1)] / [SSW / m(n-1)]
If numerator >> denominator -> Variation in data is mostly due to variations in data between actual means and less due to variation within the means. Hence there is a difference between the population means i.e there is a lower probability that the null hypothesis is correct.

F-dist = Ratio of two Chi-Squared distributions with/without different degrees of freedom.

Degrees Of Freedom: The term number of degrees of freedom means the total number of observations in the sample (= n) less the number of independent (linear) constraints or restrictions put on them.

Q) Difference between covariance and correlation?

Covariance	Correlation
Covariance is a measure to indicate the extent to which two random variables change in tandem.	Correlation is a measure used to represent how strongly two random variables are related to each other.
Covariance is nothing but a measure of correlation.	Correlation refers to the scaled form of covariance.
Covariance indicates the direction of the linear relationship between variables.	Correlation on the other hand measures both the strength and direction of the linear relationship between two variables.
Covariance can vary between -∞ and +∞	Correlation ranges between -1 and +1
Covariance is affected by the change in scale. If all the values of one variable are multiplied by a constant and all the values of another variable are multiplied, by a similar or different constant, then the covariance is changed.	Correlation is not influenced by the change in scale.
Covariance assumes the units from the product of the units of the two variables.	Correlation is dimensionless, i.e. It's a unit-free measure of the relationship between variables.

Q) Difference between regression and correlation

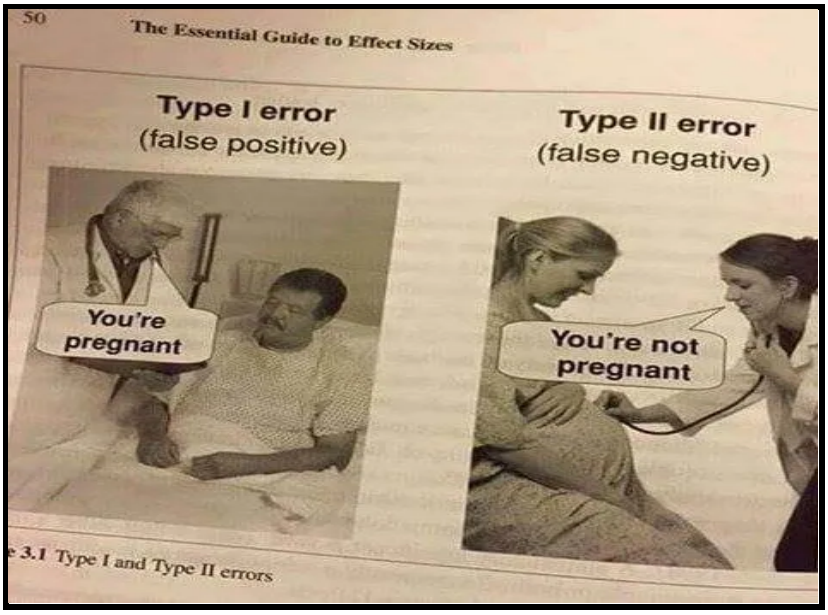
- Correlation Analysis: Primary objective is to measure the degree of linear association between two variables.
- Regression Analysis : We try to estimate or predict the average value of one var on the basis of fixed values of other variables.

14. Metrics for ML model performance - Classification

Monitoring only the ‘accuracy score’ gives an incomplete picture of your model’s performance and can impact the effectiveness.
The computed accuracy from the above model turned out to be 94%, which sounds pretty good. But, it **doesn’t reveal much information about how well the model actually did in predicting the 1’s and 0’s independently**. Nor does it say how well it would have performed with a **different prediction probability cutoff**. Let’s have a deeper look into this, starting with the confusion matrix.

Type 1 Error: False Positive: Predicted positive but actually negative
Type 2 Error: False Negative: Predicted negative but actually positive

For medical use case type 2 errors are more dangerous than type 1 errors.
Example: The covid-19 test report is negative but actually the patient is positive.



1. Accuracy = Correctly Classified / Total Cases = (TP + TN) / (TP + TN + FP + FN)
2. Sensitivity / Recall / True Positive Rate (TPR) / Bads Captured (in loan default)
= Percentage of correctly classified positives (1s)
= Percentage of actual 1s that were correctly predicted
= TP / P
= TP / TP + FN

Sensitivity matters more when **classifying the 1’s correctly is more important** than classifying the 0’s.

Example: Just like what we need here in the **BreastCancer** case, where you don't want to miss out any malignant to be classified as 'benign'.
Example: For predicting **loan default**, we don't want to miss out any bad loans (1s) to be classified as good (0s).

3. Specificity / True Negative Rate / Good Captured (in loan default)
= Percentage of correctly classified negatives (0s)
= Percentage of actual 0s that were correctly predicted
= TN / N
= TN / TN + FP

Specificity matters more when **classifying the 0's correctly is more important** than classifying the 1's.
Example: Maximizing specificity is more relevant in cases like **spam detection**, where you strictly don't want genuine messages (0's) to end up in spam (1's).

4. Precision

= What percentage of model's positive (1s) predictions are accurate
= How many positively classified were actually positive
= TP / Predicted Positive
= TP / TP + FP

5. F1-Score

= Harmonic Mean of Precision and Recall (or sensitivity / TPR)
= 2 * Precision * Recall / Precision + Recall

A high precision score gives more confidence to the model's capability to classify 1's. Combining this with Recall gives an idea of how many of the total 1's it was able to cover.
A good model should have a good precision as well as a high recall. So ideally, I want to have a measure that combines both these aspects in one single metric – the F1 Score.

6. Cohen's Kappa
classification - Cohen's kappa in plain English - Cross Validated (stackexchange.com)

Cohen's Kappa statistic is a very useful, but under-utilised, metric. Sometimes in machine learning we are faced with a **multi-class classification problem**. In those cases, measures such as the accuracy, or precision/recall do not provide the complete picture of the performance of our classifier.
In some other cases we might face a problem with **imbalanced classes**. E.g. we have two classes, say A and B, and A shows up on 5% of the time. Accuracy can be misleading, so we go for measures such as precision and recall. There are ways to combine the two, such as the F-measure, but the **F-measure does not have a very good intuitive explanation**, other than it being the harmonic mean of precision and recall.

Kappa = (Observed accuracy - Expected Accuracy) / (1 - Expected Accuracy)

The Kappa statistic (or value) is a metric that compares an Observed Accuracy with an Expected Accuracy (random chance). The kappa statistic is used not only to evaluate a single classifier, but also to **evaluate classifiers amongst themselves**. In addition, it takes into account random chance (agreement with a random classifier), which generally means it is less misleading than simply using accuracy as a metric **(an Observed Accuracy of 80% is a lot less impressive with an Expected Accuracy of 75% versus an Expected Accuracy of 50%)**.

Example:

	Cats	Dogs (Ground Truth)
Cats	10	7
Dogs	5	8

Assume that a model was built using supervised machine learning on labeled data. This doesn't always have to be the case; t**he kappa statistic is often used as a measure of reliability between two human raters**.

Observed accuracy = 0.6 = (10 + 8) / 30

Expected accuracy = This value is defined as the accuracy that any random classifier would be expected to achieve based on the confusion matrix
= ((15 * 17 / 30) + (15 * 13 / 30)) / 30 = 0.50
(For all balanced dataset, with equal number of grounds truths in all class, expected accuracy = 0.50)

Not only can this kappa statistic shed light into how the classifier itself performed, the kappa statistic for one model is directly comparable to the kappa statistic for any other model used for the same classification task.

So, in answer to your question about a 0.40 kappa, it depends. If nothing else, it means that the classifier achieved a rate of classification 2/5 of the way between whatever the expected accuracy was and 100% accuracy. If expected accuracy was 80%, that means that the classifier performed 40% (because kappa is 0.4) of 20% (because this is the distance between 80% and 100%) above 80% (because this is a kappa of 0, or random chance), or 88%. So, in that case, each increase in kappa of 0.10 indicates a 2% increase in classification accuracy. If accuracy was instead 50%, a kappa of 0.4 would mean that the classifier performed with an accuracy that is 40% (kappa of 0.4) of 50% (distance between 50% and 100%) greater than 50% (because this is a kappa of 0, or random chance), or 70%. Again, in this case that means that an increase in kappa of 0.1 indicates a 5% increase in classification accuracy.

7. AUC ROC:

Plot of TPR vs FPR or Sensitivity vs 1 - Specificity

Tells us how well can our model distinguish between the two classes for different threshold values. They can help us to find the best threshold.

Often, choosing the best model is sort of a **balance** between predicting the one's accurately or the zeroes accurately. In other words **sensitivity and specificity.**
But it would be great to have something that captures both these aspects in one single metric. This is nicely captured by the 'Receiver Operating Characteristics' curve, also called as the ROC curve. In fact, the **area under the ROC curve** can be used as an evaluation metric to compare the efficacy of the models.

The 45 degree line shows TPR = FPR. Any point on this line shows proportion of correctly classified +ve sample = Incorrectly classified negative sample
So, if we trace the curve from bottom left, the value of probability cutoff (threshold) decreases from 1 towards 0.

8. Gini Coefficient
Gini Coefficient is an indicator of how well the model outperforms random predictions. It can be computed from the area under the ROC curve using the following formula:

Gini Coefficient = (2 * AUROC) - 1

15. What's the Bias-Variance trade-off ?

The prediction error for any machine learning algorithm can be broken down into three parts:

a. Bias Error
b. Variance Error
c. Irreducible Error: It is a measure of the amount of **noise** in our data. Here it is important to understand that no matter how good we make our model, our data will have a certain amount of noise or irreducible error that can not be removed.

a. Bias are the simplifying assumptions made by a model to make the target function easier to learn.
   Generally, linear algorithms have a high bias making them fast to learn and easier to understand but generally less flexible. In turn, they have lower predictive performance on complex problems that fail to meet the simplifying assumptions of the algorithms bias.
   Examples of low-bias machine learning algorithms include: Decision Trees, k-Nearest Neighbors and Support Vector Machines.
   Examples of high-bias machine learning algorithms include: Linear Regression, Linear Discriminant Analysis and Logistic Regression.

   Model with high bias pays very little attention to the training data and oversimplifies the model. It always leads to high error on training and test data.
   f'(X) = Model
   Y = f(X) + e, Y = target variable
   Bias[f'(X)] = E[f'(X) – f(X)]
   In the simplest terms, Bias is the difference between the Predicted Value and the Expected Value.

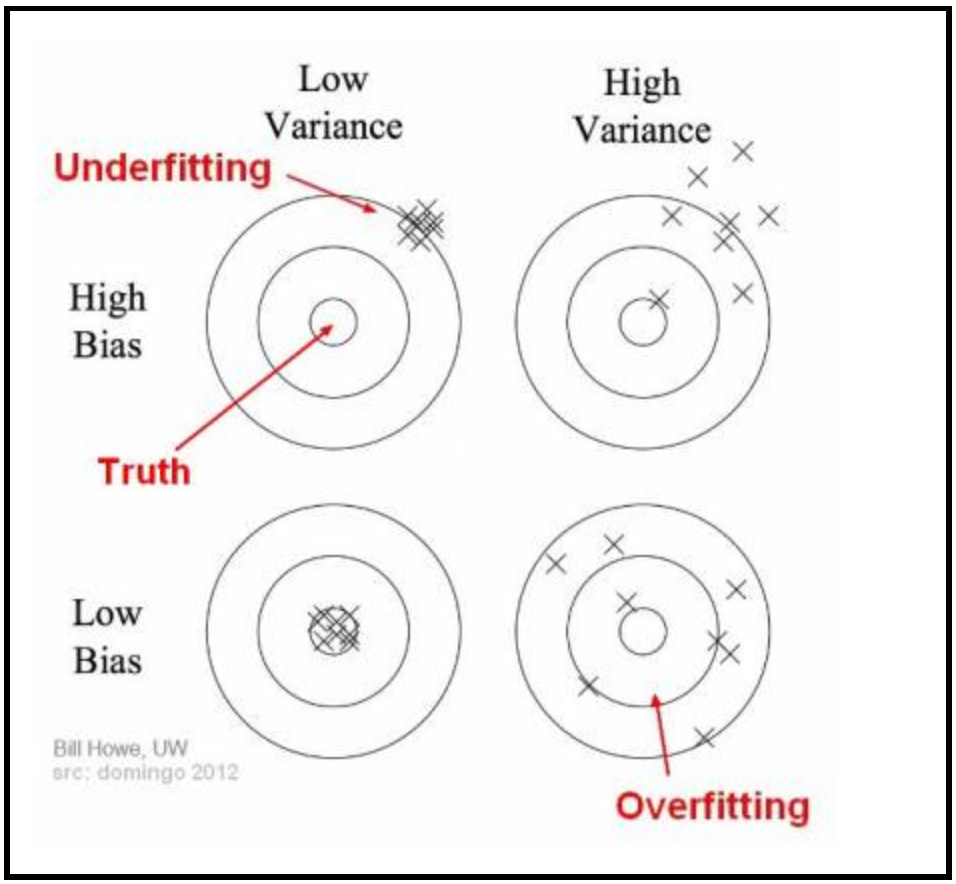
b. Machine learning algorithms that have a high variance are strongly influenced by the specifics of the training data. Model with high variance pays a lot of attention to training data and does not generalize on the data which it hasn't seen before. As a result, such models perform very well on training data but have high error rates on test data.

Examples of low-variance machine learning algorithms include: Linear Regression, Linear Discriminant Analysis and Logistic Regression.

Examples of high-variance machine learning algorithms include: Decision Trees, k-Nearest Neighbors and Support Vector Machines.

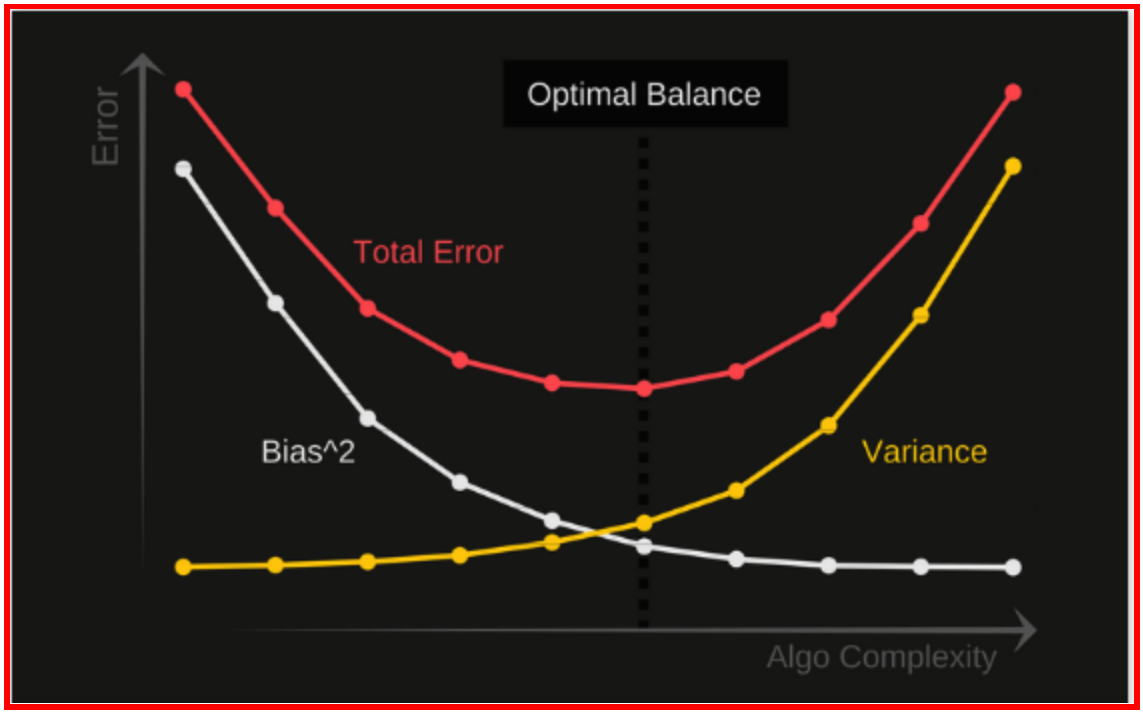
Variance[f(x))=E[X^2]−E[X]^2

The goal of any supervised machine learning algorithm is to achieve low bias and low variance. In turn the algorithm should achieve good prediction performance.

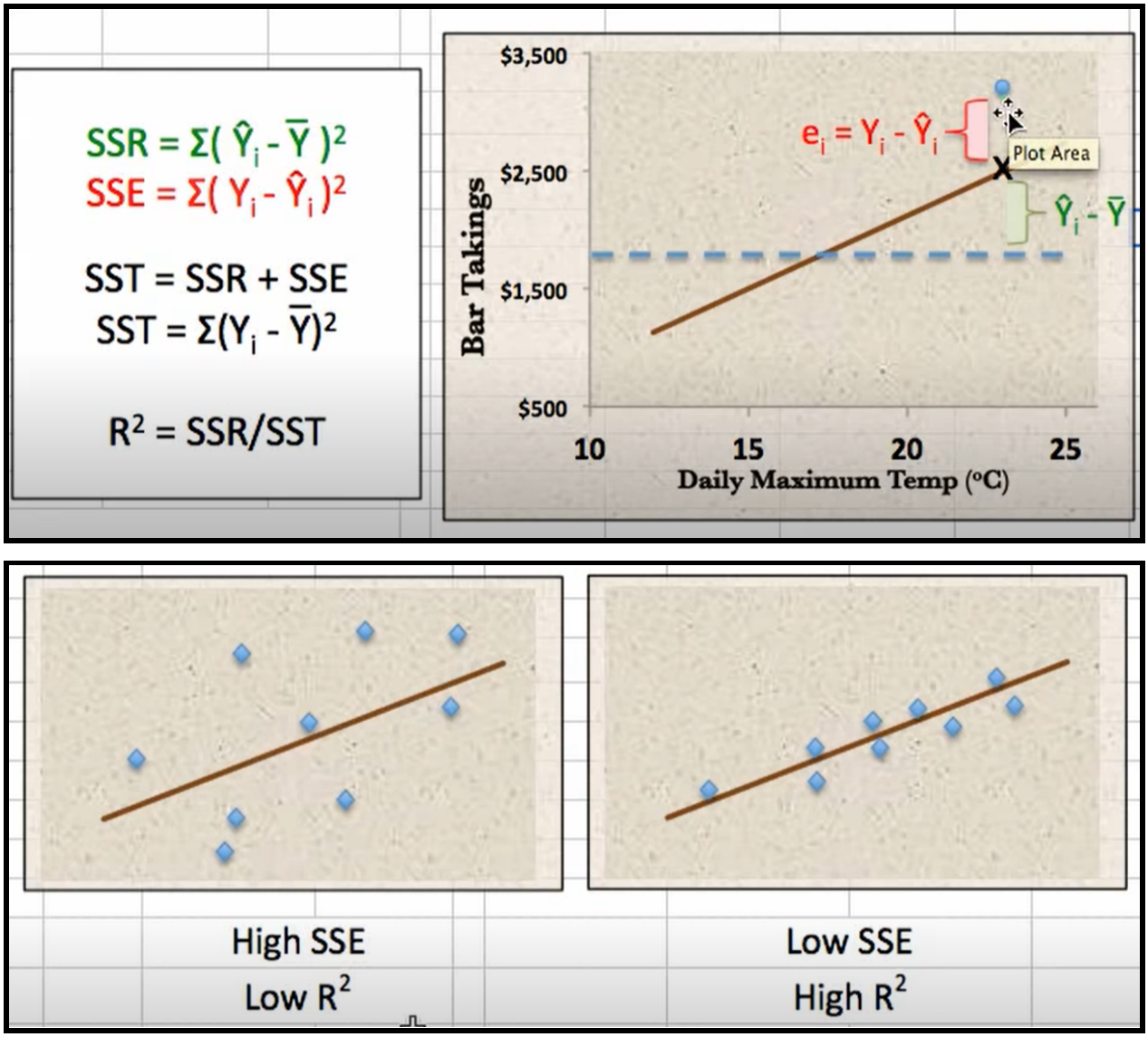


Underfitting: model **unable to capture the underlying pattern** of the data. These models usually have high bias and low variance. It happens when we have very little data to build an accurate model or when we try to build a linear model with nonlinear data. Also, these kinds of models are **very simple** to capture the complex patterns in data like Linear and logistic regression.

In supervised learning, **overfitting** happens when our model **captures the noise** along with the underlying pattern in data. It happens when we train our model a lot over noisy dataset. These models have low bias and high variance. These models are very complex like Decision trees which are prone to overfitting.



Jargons of Linear Regression: SSR (SS, Regression) , SSE (SS, Error/Residual), SST (SS, Total) , R2



R2 (coefficient of determination) = Proportion of Variation in Y that can be explained by variation in X. R-squared is a goodness-of-fit measure for linear regression models.

R-squared has Limitations

You cannot use R-squared to determine whether the coefficient estimates and predictions are biased, which is why you must assess the residual plots.

R-squared does not indicate if a regression model provides an adequate fit to your data. A good model can have a low R2 value. On the other hand, a biased model can have a high R2 value!

Some fields of study have an inherently greater amount of unexplainable variation. In these areas, your R2 values are bound to be lower. For example, studies that try to

explain human behavior generally have R2 values less than 50%. People are just harder to predict than things like physical processes.

Fortunately, if you have a **low R-squared value** but the independent variables are statistically significant, you can still draw important conclusions about the relationships between the variables. Statistically significant coefficients continue to represent the mean change in the dependent variable given a one-unit shift in the independent variable. Clearly, being able to draw conclusions like this is vital.

Are High R-squared Values Always Great? The data in the fitted line plot follow a very low noise relationship, and the R-squared is 98.5%, which seems fantastic. However, the regression line consistently under and over-predicts the data along the curve, which is bias. The Residuals versus Fits plot emphasizes this unwanted pattern. An unbiased model has residuals that are randomly scattered around zero. **Non-random residual patterns indicate a bad fit despite a high R2**. Always check your residual plots!

Even overfitting can cause high R2 values.

At first glance, R-squared seems like an easy to understand statistic that indicates how well a regression model fits a data set. However, it doesn't tell us the entire story. To get the full picture, you must **consider R2 values in combination with residual plots, other statistics**, and in-depth knowledge of the subject area.

Q) Can R2 be negative?
Ans) Yes. When the best fit line is worse than the average/mean line, then SS,Total < SS, Residual and hence R2<0.

<u>Degrees of Freedom</u>

Q) What is the minimum number of observations required to estimate this regression?

Dof = n - k -1
N = no of observation, K = no of explanatory variables (X)

Q) How does the degree of freedom relate to R-squared?
As dof decreases (more variables added to model) R2 will ONLY increase

16. R2 and adjusted-R2?

- R-squared is always between 0 and 100%.
- 0% indicates that the model explains none of the variability of the response data around its mean. 100% indicates that the model explains all the variability of the response data around its mean.

$$R^2 = 1 - \frac{SS_{RES}}{SS_{TOT}}$$

$$\hat{R}^2 = 1 - \frac{\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2}{\sum_{i=1}^{n}(Y_i - \bar{Y})^2}$$

Note: Here SS, Res = SS, Residual or SS, Error.

Disadvantage of R2: No matter how many junk independent variables or important independent variables you add to your model, the **R-Squared value will always increase**. It will never decrease with the addition of a newly independent variable, whether it could be an impactful, non-impactful, or bad variable, so we need another way to measure equivalent R Square, which penalizes our model with any junk independent variable

- The **adjusted R-squared can be negative, but it's usually not**. It is always lower than the R-squared. Use the adjusted R-square to compare models with different numbers of predictors
- Use the **predicted R-square to determine how well the model predicts new observations** and whether the model is too complicated

$$R^2_{adjusted} = 1 - \frac{(1 - R^2)(N - 1)}{N - p - 1}$$

where

R^2 = sample R-square

p = Number of predictors

N = Total sample size.

Denominator = Degrees of Freedom

If the chosen model fits worse than a straight line, R2 can be negative

Q) When can we use R2? Regression/Classification->when?

Ans: For Regression

Inferential Statistics

Inferential statistics concern generalizing from a sample to a population. A critical part of inferential statistics involves determining how far sample statistics are likely to vary from each other and from the population parameter.

Standard Error : sigma/root(N)
Mean of population = Mean of Sample
Variance of sample = Variance of Sample/N

As you might expect, the mean of the sampling distribution of the difference between means is:

$$\mu_{M_1-M_2} = \mu_1 - \mu_2$$

$$\sigma^2_{M_1-M_2} = \sigma^2_{M_1} + \sigma^2_{M_2}$$

$$\sigma^2_{M_1-M_2} = \frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}$$

$$\sigma_{M_1-M_2} = \sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}}$$

How to choose between RMSE, MAE and R2 for regression model evaluation?

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2} \qquad \text{MAE} = \frac{1}{n}\sum_{j=1}^{n}|y_j - \hat{y}_j|$$

- **Easy to understand and interpret MAE** because it directly takes the average of offsets whereas **RMSE penalizes the higher difference more than MAE**.
- RMSE will be higher than or equal to MAE. The only case where it equals MAE is when all the differences are equal or zero
- However, even after being more complex and biased towards higher deviation, RMSE is still the default metric of many models because loss function defined in

terms of **RMSE is smoothly differentiable** and makes it easier to perform mathematical operations.

- Minimizing the squared error over a set of numbers results in finding its mean, and minimizing the absolute error results in finding its median. This is the reason why **MAE is robust to outliers** whereas RMSE is not.
- The absolute value of RMSE does not actually tell how bad a model is. It can only be used to compare across two models whereas Adjusted R^2 easily does that. For example, if a model has adjusted R^2 equal to 0.05 then it is definitely poor.
- However, if you care only about prediction accuracy then RMSE is best. It is computationally simple, easily differentiable and present as default metric for most of the models.

17. Regression Losses

- **Mean Square loss**/Quadratic loss/L2 loss - To calculate the MSE, you take the difference between your model's predictions and the ground truth, square it, and average it out across the whole dataset.
- **Advantage**: The MSE is great for ensuring that our trained model has no outlier predictions with huge errors, since the MSE puts larger weight on theses errors due to the squaring part of the function
- **Disadvantage**: If our model makes a single very bad prediction, the squaring part of the function magnifies the error.
- **Mean Absolute loss**/L1 loss - To calculate the MAE, you take the difference between your model's predictions and the ground truth, apply the absolute value to that difference, and then average it out across the whole dataset
- **Advantage** : The beauty of the MAE is that its advantage directly covers the MSE disadvantage. Since we are taking the absolute value, all of the errors will be weighted on the same linear scale. Thus, unlike the MSE, we won't be putting too much weight on our outliers and our loss function provides a generic and even measure of how well our model is performing
- **Disadvantage**: If we do in fact care about the outlier predictions of our model, then the MAE won't be as effective. The large errors coming from the outliers end up being weighted the exact same as lower errors. This might result in our model being great most of the time, but making a few very poor predictions every-so-often
- **Huber Loss** - Now we know that the MSE is great for learning outliers while the MAE is great for ignoring them. But what about something in the middle?
- The Huber Loss offers the best of both worlds by balancing the MSE and MAE together

- What this equation essentially says is: for loss values less than delta, use the MSE; for loss values greater than delta, use the MAE
- **You'll want to use the Huber loss any time you feel that you need a balance between giving outliers some weight, but not too much. For cases where outliers are very important to you, use the MSE! For cases where you don't care at all about the outliers, use the MAE!**
- There may be regression problems in which the target value has a spread of values and when predicting a large value, you may not want to punish a model as heavily as mean squared error.
- Instead, you can first calculate the natural logarithm of each of the predicted values, then calculate the mean squared error. This is called the **Mean Squared Logarithmic Error loss**, or MSLE for short.
- It has the effect of relaxing the punishing effect of large differences in large predicted values. As a loss measure, it may be more appropriate when the model is predicting unscaled quantities directly.

$$L(y, \hat{y}) = \frac{1}{N}\sum_{i=0}^{N}(\log(y_i + 1) - \log(\hat{y}_i + 1))^2$$

Q) What are the common classification losses you know about?

- **Binary Cross Entropy / Log-loss**: Cross-entropy will calculate a score that summarizes the average difference between the actual and predicted probability distributions for predicting class 1. The score is minimized and a perfect cross-entropy value is 0.
- **Categorical Cross Entropy** - **Softmax** is the only activation function recommended to use with the categorical cross-entropy loss function.

Difference between Supervised, Unsupervised and Reinforcement Learning.
What's meant by semi-supervised learning ?
How is KNN different from K-Means clustering?
How does K-means clustering work? KM-clustring vs Hierarchical clustering?

Q) What does the ROC curve represent ?
Ans) ROC AUC indicates how well the prob. from positive classes are separated from negative classes.

Q)Where should AUC-ROC be used as a metric for model evaluation ?
Ans) Can be used when the outcome is a categorical variable. For multiclass - use One vs All
ROC Curve = TPR v/s FPR or Sensitivity vs (1-Specificity) for different thresholds

Q) What are some common ways of dealing with heavily imbalanced datasets ?

A. **Downsampling**: Remove some points from majority class. Disadv: Loss of data points which may contain vital info
B. **Upsampling**: Repeat some points from minority class. Disadv: Overfitting
C. **Upsampling using Artificial/synthetic points:** Use extrapolation to create artificial points
D. **Class Weight:** Assign more weight to minority class

18. Principal Component Analysis (PCA):

A Step-by-Step Explanation of Principal Component Analysis (builtin.com)

Principal Component Analysis (PCA) - Better Explained | ML+ (machinelearningplus.com)

First it identifies the hyperplane that lies closest to the data, and then it projects the data onto it.

Principal Components Analysis (PCA) is an algorithm to transform the columns of a dataset into a **new set of features called Principal Components**. By doing this, a large chunk of the information across the full dataset is effectively **compressed** in **fewer feature columns**. Comes at the expense of accuracy, but the trick in dimensionality reduction is to trade a little accuracy for simplicity. Because smaller data sets are easier to explore and visualize and make analyzing data much easier and faster for machine learning algorithms without extraneous variables to process.
So to sum up, the idea of PCA is simple — **reduce the number of variables of a data set, while preserving as much information as possible.**

Why use PCA?

Practically PCA is used for two reasons:

1. **Dimensionality Reduction:** The information distributed across a large number of columns is transformed into principal components (PC) such that the first few PCs can explain a sizeable chunk of the total information (variance). These PCs can be used as explanatory variables in Machine Learning models.
2. **Visualize Classes:** Visualising the separation of classes (or clusters) is hard for data with more than 3 dimensions (features). With the first two PCs itself, it's usually possible to see a clear separation.

Q) Is PCA a feature selection technique?

Ans) It is not a feature selection technique. Rather, it is a feature combination technique. Because each PC is a weighted additive combination of all the columns in the original dataset.

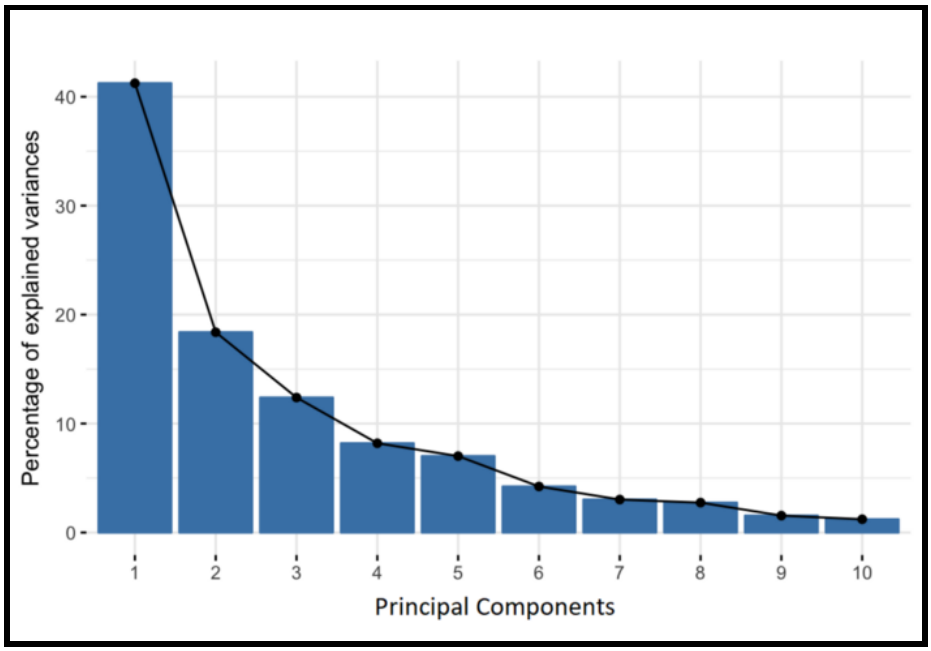
- **Create the PCs using only the X**. Later you will see, we draw a scatter plot using the first two PCs and color the points based on the actual Y. Typically, if the X's were informative enough, you should see clear clusters of points belonging to the same category.

- The key thing to understand is that, each principal component is the dot product of its weights (in pca.components_) and the mean centered data(X). What I mean by 'mean-centered' is, each column of the 'X' is subtracted from its own mean so that the mean of each column becomes zero.

$$PC_i = Weights_i \cdot X_{meancentered}$$

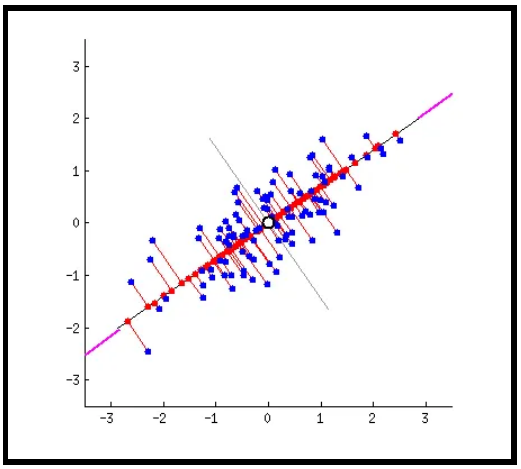
Percentage of Variance Explained with each PC (Explained variance ratio)

The PCs are usually arranged in the descending order of the variance(information) explained. PC1 contributed 22%, PC2 contributed 10% and so on. The further you go, the lesser is the contribution to the total variance.



Understanding Concepts behind PCA

To simplify things, let's imagine a dataset with only two columns. Using these two columns, I want to find a new column that better represents the 'data' contributed by these two columns.This new column can be thought of as a line that passes through these points. Such a line can be represented as a **linear combination of the two columns and explains the maximum variation present in these two columns**. In what direction do you think the line should stop so that it covers the maximum variation of the data points? **Such a line should be in a direction that minimizes the perpendicular distance of each point from the line.** It may look something like this:

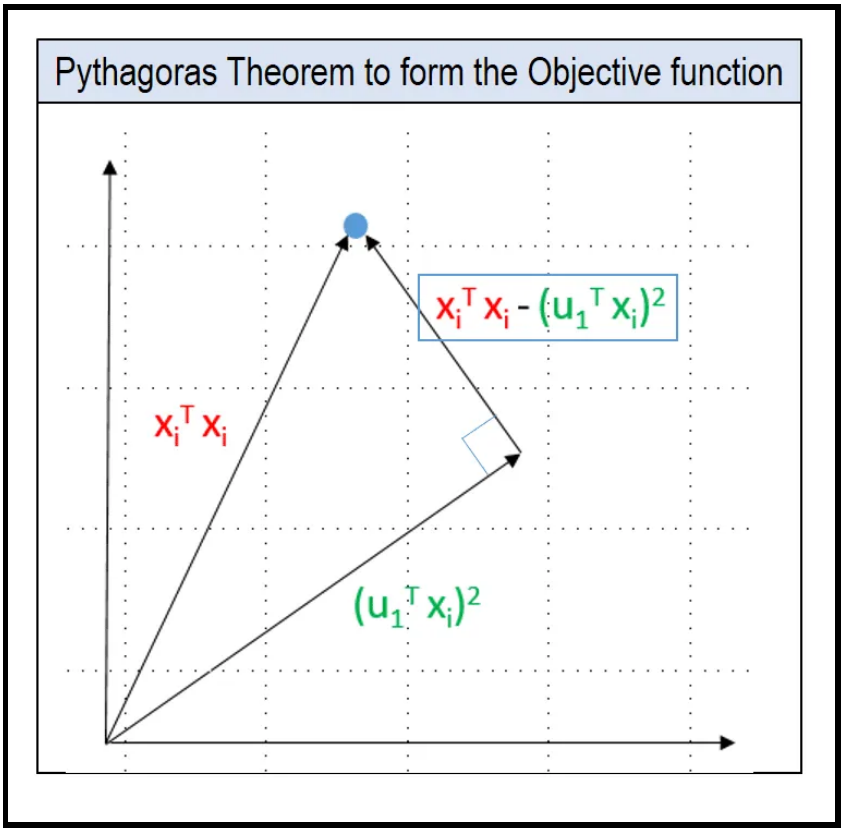


But how to determine this line?I am only interested in **determining the direction(u1)** of this line. Because, by knowing the direction u1, I can compute the projection of any point on this line. This line u1 is of length 1 unit and is called a **unit vector**. This unit vector eventually **becomes the weights of the principal components**, also called as loadings. The PCA weights (Ui) are actually unit vectors of length 1. Because, it is meant to represent only the direction.

Geometrically speaking, principal components represent the directions of the data that explain a **maximal amount of variance** (the average of the squared distances from the projected points (red dots) to the origin)**,** that is to say, the lines that capture most information of the data. The relationship between variance and information here, is that, the **larger the variance** carried by a line, the **larger the dispersion** of the data points along it, and the larger the dispersion along a line, the **more the information** it has. To put all this simply, just think of principal components as **new axes** that provide the **best angle to see and evaluate the data**, so that the differences between the observations are better visible.

Objective function:
The objective is to determine u1 so that the mean perpendicular distance from the line for all points is minimized. Here is the objective function (using Pythagoras Theorem):

$$min(\frac{1}{n}\Sigma_i^n(x_i^T x_i - (u_1^T x_i)^2))$$



It can be proved that the above equation reaches a minimum when the value of **u1** equals the **EigenVector of the covariance matrix of X**. This EigenVector is same as the PCA weights that we got earlier inside pca.components_ object. We'll see what Eigen Vectors are shortly.

Alright. But there can be a second PC to this data. The next best direction to explain the remaining variance is **perpendicular** to the first PC. Actually, there can be as many Eigen Vectors as there are columns in the dataset. And they are **orthogonal** to each other.

What is Eigen vector?

v is an eigenvector of matrix A if A(v) is a scalar multiple of v.

$$Av = \lambda v$$

Steps to Compute Principal Components from Scratch

Step 1: Standardize each column

Subtract each column by its own mean. As a result, the mean of each column becomes zero.

Q) Why Standardize?
Ans) PCA is quite sensitive regarding the variances of the initial variables. That is, if there are large differences between the ranges of initial variables, **those variables with larger ranges will dominate over those with small ranges** (For example, a variable that ranges between 0 and 100 will dominate over a variable that ranges between 0 and 1), which will lead to biased results. So, transforming the data to comparable scales can prevent this problem.

Step 2 Compute Covariance Matrix: Covariance measures how two variables are related to each other, that is, **if two variables are moving in the same direction with respect to each other or not**. When covariance is positive, it means, if one variable increases, the other increases as well. The opposite true when covariance is negative.

$$Cov(X,Y) = \frac{\Sigma(X_i - \bar{X})\,(Y_i - \bar{Y})}{n}$$

The aim of this step is to understand how the variables of the input data set are varying from the mean with respect to each other, or in other words, to **see if there is any relationship between them.** Because sometimes, variables are **highly correlated** in such a way that they contain **redundant information**. So, in order to identify these correlations, we compute the covariance matrix.
The covariance matrix is a p × p **symmetric matrix** (where p is the number of dimensions) that has as entries the **covariances associated with all possible pairs of the initial variables**. For example, for a 3-dimensional data set with 3 variables x, y, and z, the covariance matrix is a 3×3 matrix of this from:

$$\begin{bmatrix} Cov(x,x) & Cov(x,y) & Cov(x,z) \\ Cov(y,x) & Cov(y,y) & Cov(y,z) \\ Cov(z,x) & Cov(z,y) & Cov(z,z) \end{bmatrix}$$

Step 3: Compute Eigen values and Eigen Vectors
Eigenvalues (amount of variance) and Eigenvectors (directions of the axes where there is the most variance(most information) and that we call Principal Components.) represent the amount of variance explained and how the columns are related to each other. The length of Eigenvectors is one.

After having the principal components, to compute the percentage of variance (information) accounted for by each component, we divide the eigenvalue of each component by the sum of eigenvalues.

Step 4: Derive Principal Component Features by taking dot product of eigen vector (weights) and standardized columns

Disadvantage of PCA:
An important thing to realize here is that, the principal components are **less interpretable** and don't have any real meaning since they are constructed as linear combinations of the initial variables.

Incremental PCA
One problem with the preceding implementation of PCA is that it requires the whole training set to fit in memory in order for the SVD algorithm to run. Fortunately, Incremental PCA (IPCA) algorithms have been developed: you can split the training set into **mini-batches** and feed an IPCA algorithm one mini-batch at a time. This is useful for large training

Kernel PCA
In Chapter 5 we discussed the kernel trick, a mathematical technique that implicitly maps instances into a very high-dimensional space (called the feature space), enabling nonlinear classification and regression with Support Vector Machines. Recall that a linear decision boundary in the high-dimensional feature space corresponds to a complex nonlinear decision boundary in the original space. It turns out that the same trick can be applied to PCA, making it possible to perform complex nonlinear projections for dimensionality reduction. This is called Kernel PCA (kPCA).6 It is often good at preserving clusters of instances after projection, or sometimes even unrolling datasets that lie close to a twisted manifold.

SVD and PCA:
PCA and SVD explained with numpy. How exactly are principal component… | by Zichen Wang | Towards Data Science

We can actually perform PCA using SVD, or vice versa. In fact, most implementations of PCA actually perform SVD under the hood rather than doing eigen decomposition on the covariance matrix because SVD can be much more efficient and is able to handle sparse matrices. In addition, there are reduced forms of SVD which are even more economic to compute.

Q) What is the basic principle behind PCA ? What is the criteria used for reducing the dimension in PCA ?
Q) What is SVD ? What are some of its areas of applications ?

19. KNN

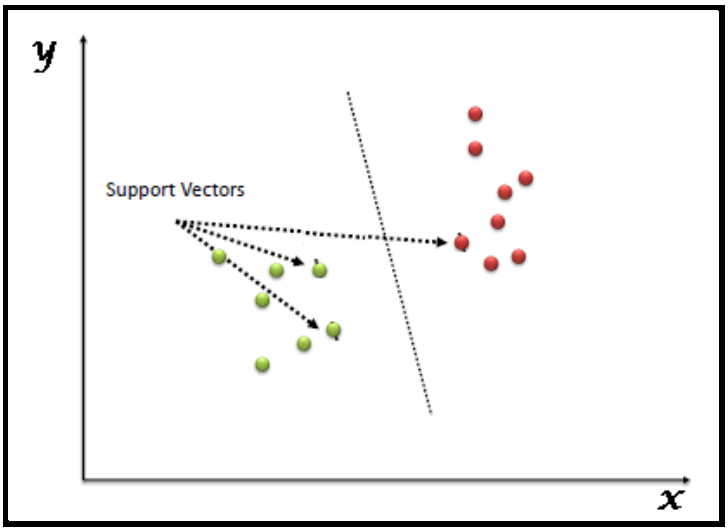
K-Nearest Neighbors for Machine Learning (machinelearningmastery.com)

- KNN stores the entire training dataset which it uses as its representation.
- KNN does not learn any model.
- KNN makes predictions just-in-time by calculating the similarity between an input sample and each training instance.
- There are many distance measures to choose from to match the structure of your input data.
- That it is a good idea to rescale your data, such as using normalization, when using KNN.

20 .SVM

SVM | Support Vector Machine Algorithm in Machine Learning (analyticsvidhya.com)

What is Support Vector Machine?
"Support Vector Machine" (SVM) is a **supervised** machine learning algorithm which can be used for both **classification** or **regression** challenges. However, it is mostly used in classification problems. In the SVM algorithm, we plot each data item as a point in n-dimensional space (where n is number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the **hyper-plane that differentiates** the two classes very well (look at the below snapshot).



Support Vectors are simply the co-ordinates of individual observation. The SVM classifier is a frontier which best segregates the two classes (hyper-plane/ line).

How does it work?

Above, we got accustomed to the process of segregating the two classes with a hyper-plane. Now the burning question is “How can we identify the right hyper-plane?”.

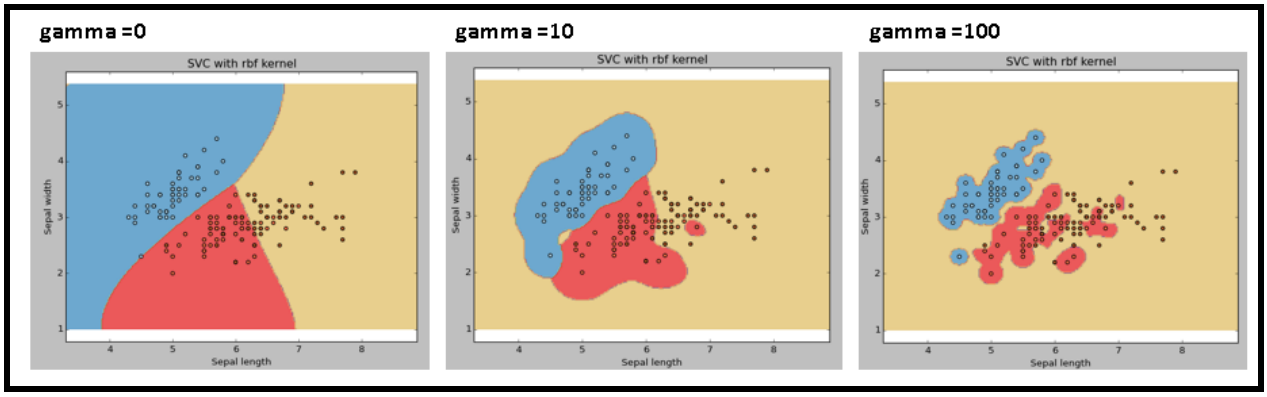
1. “Select the hyper-plane which segregates the two classes better”
2. Here, **maximizing** the distances between nearest data point (either class) and hyper-plane will help us to decide the right hyper-plane. This distance is called as **Margin.**
3. SVM selects the hyper-plane which classifies the classes **accurately prior to maximizing margin**
4. The SVM algorithm has a feature to **ignore outliers** and find the hyper-plane that has the maximum margin. Hence, we can say, SVM classification is robust to outliers.
5. In the SVM classifier, it is easy to have a linear hyper-plane between these two classes. But, another burning question which arises is, should we need to add this feature manually to have a hyper-plane. No, the SVM algorithm has a technique called the **kernel** trick. **The SVM kernel is a function that takes low dimensional input space and transforms it to a higher dimensional space** i.e. it converts not separable problem to separable problem. It is mostly useful in non-linear separation problem. Simply put, it does some extremely complex data transformations, then finds out the process to separate the data based on the labels or outputs you’ve defined.

How to tune Parameters of SVM?

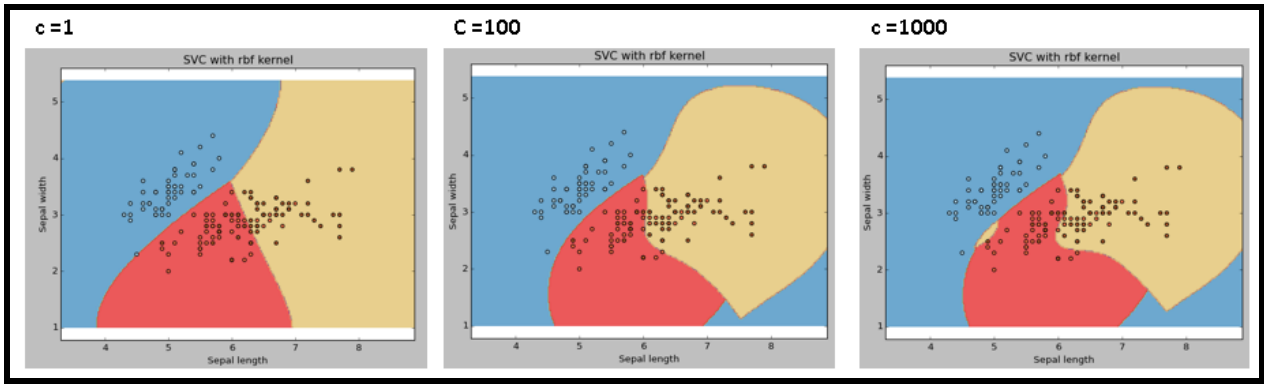
sklearn.svm.SVC(C=1.0, kernel='rbf', degree=3, gamma=0.0, coef0=0.0, shrinking=True, probability=False,tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1, random_state=None)

kernel: We have already discussed it. Here, we have various options available with kernel like, “linear”, “rbf”,”poly” and others (default value is “rbf”). Here “rbf” and “poly” are useful for non-linear hyper-plane.

gamma: Kernel coefficient for ‘rbf’, ‘poly’ and ‘sigmoid’. High value of gamma will try to exactly fit the as per training data set and cause overfitting problem.



C: Penalty parameter C of the error term. It also controls the trade-off between smooth decision boundaries and classifying the training points correctly.



Pros and Cons associated with SVM

Pros:

- It works really well with a clear margin of separation
- It is effective in **high dimensional** spaces.

- It is effective in cases where the number of dimensions is greater than the number of samples.
- It uses a **subset of training points** in the decision function (called support vectors), so it is also memory efficient.

Cons:

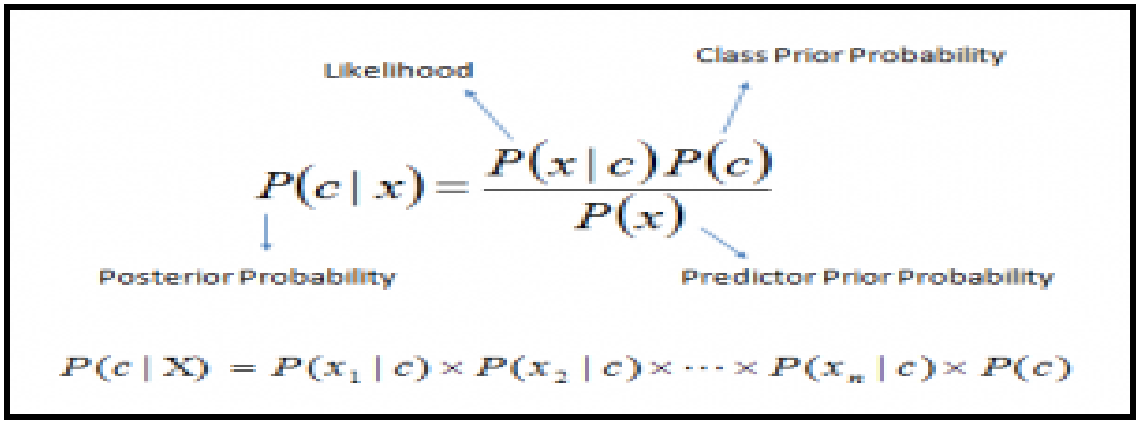
- It doesn't perform well when we have **large data** set because the required training time is higher
- It also doesn't perform very well, when the data set has more noise i.e. target **classes are overlapping**
- SVM doesn't directly provide probability estimates, these are calculated using an expensive five-fold cross-validation. It is included in the related SVC method of Python scikit-learn library.

21. NAIVE BAYES

Learn Naive Bayes Algorithm | Naive Bayes Classifier Examples (analyticsvidhya.com)

It is a **classification** technique based on Bayes' Theorem with an assumption of **independence among predictors**. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature.
Naive Bayes model is easy to build and particularly useful for **very large data sets**. Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods.

Bayes theorem provides a way of calculating posterior probability P(c|x) from P(c), P(x) and P(x|c). Look at the equation below:



- P(c|x) is the posterior probability of class (c, target) given predictor (x, attributes).
- P(c) is the prior probability of class.
- P(x|c) is the likelihood which is the probability of predictor given class.
- P(x) is the prior probability of predictor

How does Naive Bayes algorithm work?

Let's understand it using an example. Below I have a training data set of weather and corresponding target variable 'Play' (suggesting possibilities of playing). Now, we need to classify whether players will play or not based on weather condition. Let's follow the below steps to perform it.

Step 1: Convert the data set into a frequency table

Step 2: Create Likelihood table by finding the probabilities like Overcast probability = 0.29 and probability of playing is 0.64.

naive bayes, probability, example

Weather	Play
Sunny	No
Overcast	Yes
Rainy	Yes
Sunny	Yes
Sunny	Yes
Overcast	Yes
Rainy	No
Rainy	No
Sunny	Yes
Rainy	Yes
Sunny	No
Overcast	Yes
Overcast	Yes
Rainy	No

Frequency Table		
Weather	No	Yes
Overcast		4
Rainy	3	2
Sunny	2	3
Grand Total	5	9

Likelihood table				
Weather	No	Yes		
Overcast		4	=4/14	0.29
Rainy	3	2	=5/14	0.36
Sunny	2	3	=5/14	0.36
All	5	9		
	=5/14	=9/14		
	0.36	0.64		

Step 3: Now, use Naive Bayesian equation to calculate the posterior probability for each class. The class with the highest posterior probability is the outcome of prediction.

Problem: Players will play if weather is sunny. Is this statement is correct?

We can solve it using above discussed method of posterior probability.

P(Yes | Sunny) = P(Sunny | Yes) * P(Yes) / P (Sunny)

Here we have P (Sunny |Yes) = 3/9 = 0.33, P(Sunny) = 5/14 = 0.36, P(Yes)= 9/14 = 0.64

Now, P (Yes | Sunny) = 0.33 * 0.64 / 0.36 = 0.60, which has higher probability.

Naive Bayes uses a similar method to predict the probability of different classes based on various attributes. This algorithm is **mostly used in text classification and with problems having multiple classes**.

Applications of Naive Bayes Algorithms

Real time Prediction: Naive Bayes is an eager learning classifier and it is sure fast. Thus, it could be used for making predictions in real time.

Multi class Prediction: This algorithm is also well known for multi class prediction feature. Here we can predict the probability of multiple classes of target variable.

Text classification/ Spam Filtering/ Sentiment Analysis: Naive Bayes classifiers mostly used in text classification (due to better result in multi class problems and independence rule) have higher success rate as compared to other algorithms. As a

result, it is widely used in Spam filtering (identify spam e-mail) and Sentiment Analysis (in social media analysis, to identify positive and negative customer sentiments)
Recommendation System: Naive Bayes Classifier and Collaborative Filtering together builds a Recommendation System that uses machine learning and data mining techniques to filter unseen information and predict whether a user would like a given resource or not

Tips to improve the power of Naive Bayes Model
Here are some tips for improving power of Naive Bayes Model:

- If continuous features do not have normal distribution, we should use **transformation** or different methods to convert it in normal distribution.
- If test data set has **zero frequency issue**, apply smoothing techniques "**Laplace Correction**" to predict the class of test data set.
- **Remove correlated** features, as the highly correlated features are voted twice in the model and it can lead to over inflating importance.
- Naive Bayes classifiers has limited options for parameter tuning like alpha=1 for **smoothing**, **fit_prior=**[True|False] to learn class prior probabilities or not and some other options. I would recommend to focus on your pre-processing of data and the feature selection.
- You might think to apply some classifier combination techniques like ensembling, bagging and boosting but these methods would not help. Actually, "ensembling, boosting, bagging" won't help since their purpose is to reduce variance. **Naive Bayes has no variance to minimize.**

Q) When and Why to apply feature scaling / normalization?
Ans) Every feature has two components - unit & magnitude

- Calculation of Euclidean distance takes time if the difference in magnitude between two features is large. Hence, whenever an algorithm uses Euclidean distance, we need to perform feature scaling. (KNN, K-means clustering)
- Whenever Gradient Descent is used, feature scaling is performed to reach the global minima fast. (in Linear Regression, Deep Learning - ANN, CNN)

Types of normalization:

1. **Standard Scaler**: Standardize features by removing the mean and scaling to unit variance. The standard score of a sample x is calculated as: $z = (x - u) / s$

2. **Min-max scaler:** In this approach, the data is scaled to a fixed range - usually 0 to 1. The cost of having this bounded range - in contrast to standardization - is that we will end up with **smaller standard deviations**, which can **suppress the effect of outliers**.A Min-Max scaling is typically done via the following equation:

   Xsc = X−Xmin / Xmax−Xmin

Note: Not required in Decision Tree based algorithms

Q) How to select the best model?
Ans) Use k-fold Cross Validation to find the performance of the model and compare this with other models to find the best model - giving the best k-fold cv score.
OR
AIC / BIC / MDL
OR
Train-Val-Test Split

22. CLUSTERING

The 5 Clustering Algorithms Data Scientists Need to Know | by George Seif | Towards Data Science

Clustering | Types Of Clustering | Clustering Applications (analyticsvidhya.com)

Clustering in Python | What is K means Clustering? (analyticsvidhya.com)

40 Questions to test Data Scientist on Clustering Techniques (Skilltest Solution) (analyticsvidhya.com)

23. CLASS IMBALANCE

8 Tactics to Combat Imbalanced Classes in Your Machine Learning Dataset (machinelearningmastery.com)

Class Imbalance | Handling Imbalanced Data Using Python (analyticsvidhya.com)

1) Can You Collect More Data?

2) Try Changing Your Performance Metric

3) Try Resampling Your Dataset

4) Try Generate Synthetic Samples

5) Try Different Algorithms

6) Try Penalized Models

7) Try a Different Perspective

8) Get Creative

SQL:

https://sqlzoo.net/
http://thedatamonk.com/
https://www.hackerrank.com/domains/sql?badge_type=sql
https://www.w3schools.com/sql/default.asp

1. Difference between UNIQUE (all values in col are diff), DISTINCT (removes duplicate records) and DIFFERENT
2. When to use WHERE , HAVING (with groupby objects)
3. LPAD, RPAD
4. INSERT INTO (old table) and SELECT INTO (new table)
5. IFNULL, ISNULL
6. Create PROCEDURES
7. SELF JOIN and other JOINS
8. Wildcards
9. Aliases
10. ANY, ALL
11. EXISTS
12. CASE
13. UNION
14. Primary key, unique key and foreign key.
15. Windows Function

REGULARIZATION

FEATURE ENGINEERING

FEATURE SELECTION

TIME SERIES

BOOK: Hands on ML with Scikit Learn and Tensorflow

https://www.amazon.in/Hands-Machine-Learning-Scikit-Learn-TensorFlow/dp/1491962291

Batch Learning (Offline Learning) - train on entire data and use the model. Update model offline, after certain intervals of time - incapable of learning incrementally
vs
Online Learning (Mini-Batches) - on the fly - fast and cheap - requires less computation resources

Main Challenges of Machine Learning:
Insufficient Quantity of Training Data
Nonrepresentative Training Data
Poor-Quality Data
Irrelevant Features

Overfitting:

Overgeneralizing - it means that the model performs well on the training data, but it does not generalize well - detect patterns in the noise itself.
The possible solutions are:
• To **simplify** the model by selecting one with fewer parameters (e.g., a linear model rather than a high-degree polynomial model), by reducing the number of attributes in the training data or by constraining the model
• To gather **more training data**
• To **reduce the noise** in the training data (e.g., fix data errors and remove outliers)

Constraining a model to make it simpler and reduce the risk of overfitting is called **regularization**.

A **hyperparameter** is a parameter of a learning algorithm (not of the model). As such, it is not affected by the learning algorithm itself; it must be **set prior to training** and remains constant during training.

You train multiple models with various hyperparameters using the training set, you select the model and hyperparameters that perform best on the **validation** set, and when you're happy with your model you run a **single final test** against the test set to get an estimate of the generalization error.

To avoid "wasting" too much training data in validation sets, a common technique is to use **cross-validation**: the training set is split into complementary subsets, and each model is trained against a different combination of these subsets and validated against the remaining parts

If the dataset is large enough, train_test_split works fine, else we need to use **StratifiedShuffleSplit** to ensure that the train/test set is representative of the entire dataset.

The F1 score is the harmonic mean of precision and recall (Equation 3-3). Whereas the regular mean treats all values equally, the harmonic mean gives much more weight to low values. As a result, the classifier will only get a **high F1 score if both recall and precision are high.**

Precision-Recall Curve v/s ROC Curve

The receiver operating characteristic (ROC) curve is another common tool used with binary classifiers. It is very similar to the precision/recall curve, but instead of plotting precision versus recall, the ROC curve plots the true positive rate (another name for recall) against the false positive rate.

Since the ROC curve is so similar to the precision/recall (or PR) curve, you may wonder how to decide which one to use. As a rule of thumb, you should **prefer the PR curve whenever the positive class is rare or when you care more about the false positives than the false negatives,** and the ROC curve otherwise. For example, looking at the previous ROC curve (and the ROC AUC score), you may think that the classifier is really good. But this is mostly because there are few positives (5s) compared to the negatives (non-5s). In contrast, the PR curve makes it clear that the classifier has room for improvement (the curve could be closer to the topright corner).

PR Curve: When imbalanced dataset
ROC: Balanced Dataset

Another difference: Use of **true negatives** in the False Positive Rate in the ROC Curve and the careful avoidance of this rate in the Precision-Recall curve.

Chapter 4:Training Models- Questions

1. What Linear Regression training algorithm can you use if you have a training set with millions of features?
2. Suppose the features in your training set have very different scales. What algorithms might suffer from this, and how? What can you do about it?
3. Can Gradient Descent get stuck in a local minimum when training a Logistic Regression model?
4. Do all Gradient Descent algorithms lead to the same model provided you let them run long enough?
5. Suppose you use Batch Gradient Descent and you plot the validation error at every epoch. If you notice that the validation error consistently goes up, what is likely going on? How can you fix this?
6. Is it a good idea to stop Mini-batch Gradient Descent immediately when the validation error goes up?
7. Which Gradient Descent algorithm (among those we discussed) will reach the vicinity of the optimal solution the fastest? Which will actually converge? How can you make the others converge as well?
8. Suppose you are using Polynomial Regression. You plot the learning curves and you notice that there is a large gap between the training error and the validation error. What is happening? What are three ways to solve this?
9. Suppose you are using Ridge Regression and you notice that the training error and the validation error are almost equal and fairly high. Would you say that the model suffers from high bias or high variance? Should you increase the regularization hyperparameter α or reduce it?
10. Why would you want to use:
• Ridge Regression instead of Linear Regression?
• Lasso instead of Ridge Regression?
• Elastic Net instead of Lasso?
11. Suppose you want to classify pictures as outdoor/indoor and daytime/nighttime. Should you implement two Logistic Regression classifiers or one Softmax Regression classifier?

Chapter 4: Training Linear Models-Answers

1. If you have a training set with millions of features you can use Stochastic Gradient Descent or Mini-batch Gradient Descent, and perhaps Batch Gradient Descent if the training set fits in memory. But you cannot use the Normal Equation because the computational complexity grows quickly (more than quadratically) with the number of features.
2. If the features in your training set have very different scales, the cost function will have the shape of an elongated bowl, so the Gradient Descent algorithms will take a long time to converge. To solve this you should scale the data before training the model. Note that the Normal Equation will work just fine without scaling.
3. Gradient Descent cannot get stuck in a local minimum when training a Logistic Regression model because the cost function is convex.
4. If the optimization problem is convex (such as Linear Regression or Logistic Regression), and assuming the learning rate is not too high, then all Gradient Descent algorithms will approach the global optimum and end up producing fairly similar models. However, unless you gradually reduce the learning rate, Stochastic GD and Mini-batch GD will never truly converge; instead, they will keep jumping back and forth around the global optimum. This means that even if you let them run for a very long time, these Gradient Descent algorithms will produce slightly different models.
5. If the validation error consistently goes up after every epoch, then one possibility is that the learning rate is too high and the algorithm is diverging. If the training error also goes up, then this is clearly the problem and you should reduce the learning rate. However, if the training error is not going up, then your model is overfitting the training set and you should stop training.
6. Due to their random nature, neither Stochastic Gradient Descent nor Mini-batch Gradient Descent is guaranteed to make progress at every single training iteration.
So if you immediately stop training when the validation error goes up, you may stop much too early, before the optimum is reached. A better option is to save the model at regular intervals, and when it has not improved for a long time (meaning it will probably never beat the record), you can revert to the best saved model.
7. Stochastic Gradient Descent has the fastest training iteration since it considers only one training instance at a time, so it is generally the first to reach the vicinity of the global optimum (or Mini-batch GD with a very small mini-batch size). However, only Batch Gradient Descent will actually converge, given enough training time. As mentioned, Stochastic GD and Mini-batch GD will bounce around the optimum, unless you gradually reduce the learning rate.
8. If the validation error is much higher than the training error, this is likely because your model is overfitting the training set. One way to try to fix this is to reduce the polynomial degree: a model with fewer degrees of freedom is less likely to overfit. Another thing

you can try is to regularize the model—for example, by adding an ℓ2 penalty (Ridge) or an ℓ1 penalty (Lasso) to the cost function. This will also reduce the degrees of freedom of the model. Lastly, you can try to increase the size of the training set.
9. If both the training error and the validation error are almost equal and fairly high, the model is likely underfitting the training set, which means it has a high bias. You should try reducing the regularization hyperparameter α.
10. Let's see:
• A model with some regularization typically performs better than a model without any regularization, so you should generally prefer Ridge Regression over plain Linear Regression.
• Lasso Regression uses an ℓ1 penalty, which tends to push the weights down to exactly zero. This leads to sparse models, where all weights are zero except for the most important weights. This is a way to perform feature selection automatically, which is good if you suspect that only a few features actually matter. When you are not sure, you should prefer Ridge Regression.
• Elastic Net is generally preferred over Lasso since Lasso may behave erratically in some cases (when several features are strongly correlated or when there are more features than training instances). However, it does add an extra hyperparameter to tune. If you just want Lasso without the erratic behavior, you can just use Elastic Net with an l1_ratio close to 1.
11. If you want to classify pictures as outdoor/indoor and daytime/nighttime, since these are not exclusive classes (i.e., all four combinations are possible) you should train two Logistic Regression classifiers.

So should you use Gini impurity or entropy? The truth is, most of the time it does not make a big difference: they lead to similar trees. Gini impurity is slightly faster to compute, so it is a good default. However, when they differ, Gini impurity tends to isolate the most frequent class in its own branch of the tree, while entropy tends to produce slightly more balanced trees.

Dimensionality Reduction - Curse of Dimensionality

Reducing dimensionality does lose some information (just like compressing an image to JPEG can degrade its quality), so even though it will **speed up training**, it may also make your system perform slightly worse. It also makes your pipelines a bit more complex and thus harder to maintain. So you should first try to train your system with the original data before considering using dimensionality reduction if training is too slow. In some cases, however, reducing the dimensionality of the training data may filter out

some noise and unnecessary details and thus result in **higher performance** (but in general it won't; it will just speed up training).
Apart from speeding up training, dimensionality reduction is also extremely useful for **data visualization**.

This fact implies that high dimensional datasets are at risk of being very **sparse**: most training instances are likely to be far away from each other. Of course, this also means that a new instance will likely be far away from any training instance, making predictions much less reliable than in lower dimensions, since they will be based on much larger extrapolations. In short, **the more dimensions t**he training set has, the greater the risk of **overfitting** it.
In theory, one solution to the curse of dimensionality could be to **increase the size of the training set** to reach a sufficient density of training instances. Unfortunately, in practice, the number of training instances required to reach a given density grows exponentially with the number of dimensions.

Main Approaches for Dimensionality Reduction
Before we dive into specific dimensionality reduction algorithms, let's take a look at the two main approaches to reducing dimensionality: **projection and Manifold Learning.**

Projection
In most real-world problems, training instances are not spread out uniformly across all dimensions. Many features are almost constant, while others are highly correlated (as discussed earlier for MNIST). As a result, all training instances actually lie within (or close to) a much lower-dimensional subspace of the high-dimensional space.

Manifold Learning
Many dimensionality reduction algorithms work by modeling the manifold on which the training instances lie; this is called Manifold Learning. It relies on the manifold assumption, also called the manifold hypothesis, which holds that most real-world high-dimensional datasets lie close to a much dimensional manifold. This assumption is very often empirically observed.
Once again, think about the **MNIST** dataset: all handwritten digit images have some similarities. They are made of connected lines, the borders are white, they are more or less centered, and so on. If you randomly generated images, only a ridiculously tiny fraction of them would look like handwritten digits. **In other words, the degrees of freedom available to you if you try to create a digit image are dramatically lower than the degrees of freedom you would have if you were allowed to generate any**

image you wanted. These constraints tend to squeeze the dataset into a lower dimensional manifold.

Exercises

1. What are the main motivations for reducing a dataset's dimensionality? What are the main drawbacks?
2. What is the curse of dimensionality?
3. Once a dataset's dimensionality has been reduced, is it possible to reverse the operation? If so, how? If not, why?
4. Can PCA be used to reduce the dimensionality of a highly nonlinear dataset?
5. Suppose you perform PCA on a 1,000-dimensional dataset, setting the explained variance ratio to 95%. How many dimensions will the resulting dataset have?
6. In what cases would you use vanilla PCA, Incremental PCA, Randomized PCA, or Kernel PCA?
7. How can you evaluate the performance of a dimensionality reduction algorithm on your dataset?
8. Does it make any sense to chain two different dimensionality reduction algorithms?

1. Motivations and drawbacks: • The main motivations for dimensionality reduction are: — To speed up a subsequent training algorithm (in some cases it may even remove noise and redundant features, making the training algorithm per- form better). — To visualize the data and gain insights on the most important features. — Simply to save space (compression). • The main drawbacks are: — Some information is lost, possibly degrading the performance of subse- quent training algorithms. — It can be computationally intensive. — It adds some complexity to your Machine Learning pipelines. — Transformed features are often hard to interpret.
2. The curse of dimensionality refers to the fact that many problems that do not exist in low-dimensional space arise in high-dimensional space. In Machine Learning, one common manifestation is the fact that randomly sampled highdimensional vectors are generally very sparse, increasing the risk of overfitting and making it very difficult to identify patterns in the data without having plenty of training data.
3. Once a dataset's dimensionality has been reduced using one of the algorithms we discussed, it is almost always impossible to perfectly reverse the operation, because some information gets lost during dimensionality reduction. Moreover, while some algorithms (such as PCA) have a simple reverse transformation procedure that can reconstruct a dataset relatively similar to the original, other algorithms (such as T-SNE) do not.
4. PCA can be used to significantly reduce the dimensionality of most datasets, even if they are highly nonlinear, because it can at least get rid of useless dimensions.

However, if there are no useless dimensions—for example, the Swiss roll—then reducing dimensionality with PCA will lose too much information. You want to unroll the Swiss roll, not squash it.

5. That's a trick question: it depends on the dataset. Let's look at two extreme examples. First, suppose the dataset is composed of points that are almost perfectly aligned. In this case, PCA can reduce the dataset down to just one dimension while still preserving 95% of the variance. Now imagine that the dataset is composed of perfectly random points, scattered all around the 1,000 dimensions. In this case all 1,000 dimensions are required to preserve 95% of the variance. So the answer is, it depends on the dataset, and it could be any number between 1 and 1,000. Plotting the explained variance as a function of the number of dimensions is one way to get a rough idea of the dataset's intrinsic dimensionality.

6. Regular PCA is the default, but it works only if the dataset fits in memory. Incremental PCA is useful for large datasets that don't fit in memory, but it is slower than regular PCA, so if the dataset fits in memory you should prefer regular PCA. Incremental PCA is also useful for online tasks, when you need to apply PCA on the fly, every time a new instance arrives. Randomized PCA is useful when you want to considerably reduce dimensionality and the dataset fits in memory; in this case, it is much faster than regular PCA. Finally, Kernel PCA is useful for nonlinear datasets.

7. Intuitively, a dimensionality reduction algorithm performs well if it eliminates a lot of dimensions from the dataset without losing too much information. One way to measure this is to apply the reverse transformation and measure the reconstruction error. However, not all dimensionality reduction algorithms provide a reverse transformation. Alternatively, if you are using dimensionality reduction as a preprocessing step before another Machine Learning algorithm (e.g., a Random Forest classifier), then you can simply measure the performance of that second algorithm; if dimensionality reduction did not lose too much information, then the algorithm should perform just as well as when using the original dataset.

8. It can absolutely make sense to chain two different dimensionality reduction algorithms. A common example is using PCA to quickly get rid of a large number of useless dimensions, then applying another much slower dimensionality reduction algorithm, such as LLE. This two-step approach will likely yield the same performance as using LLE only, but in a fraction of the time.

26. TABLEAU

https://www.youtube.com/playlist?list=PLWPirh4EWFpGXTBu8ldLZGJCUeTMBpJFK

Dimensions: Text, Date, Geographical info
Measures: Numbers

Connecting to different databases

Live vs Extract

6 Data Types:

1. Numbers (decimal) and Numbers (whole)
2. Date and Time
3. Date
4. String
5. Boolean
6. Geographic Data Types

Drill Down and Hierarchies

Discrete - Blue colour
Continuous - Green colour

Parameters (Eg: Top N) and Sets
Grouping
Filters

Scatter Plot
Line Graph
Reference lines
Bubble Chart
Bar and Stacked chart
Tree Map
Bump Chart: Line chart where rank changes with time
Funnel Chart
Waterfall (Gnan Chart)
Piechart

Maps

27. Miscellaneous Questions:

1.Suppose your dataset has a lot of missing values. How would you address this issue?
2.What kind of imputation can be used in case of missing categorical values and missing continuous values?
3.Suppose your features are highly skewed (either left or right). Would this affect the model training ? If yes, how to address the issue ?
4.What are outliers ? What are some common techniques used to detect outliers in the data ?
5.Suppose Pearson correlation between V1 and V2 is zero. In such a case, is it right to conclude that V1 and V2 do not have any relation between them?
6.What are some forms of Data Leakages ? How can they be prevented ?
7.What is prior probability and posterior probability ?
8.Can normal stratified K-fold cross validation be used for a time series ? Why ?
9.If not, what cross validation technique can be used as an alternative ?
10. Differentiate between a model's parameters and hyperparameters.
11. Differentiate between factor analysis and cluster analysis.
12. You are provided with a very big dataset which goes into the order of millions of rows and thousands of features and you have a lack of computing resources to process the entire dataset at once. How would you approach this problem?
13. Differentiate between K-Means Clustering and Hierarchical clustering.
14. What is A/B Testing in ML?
15. Given a dataset, how would you apply KNN on it ?
16. Feature Engineering techniques
17.. What is the loss function used in Random Forest ?

28. SELF PROJECTS:

https://github.com/ashishpatel26/500-AI-Machine-learning-Deep-learning-Computer-vision-NLP-Projects-with-code

29. COURSERA Deep Learning Specialization Notes:

https://github.com/mbadry1/DeepLearning.ai-Summary

30. DECISION TREES

1. What is a Decision Tree ? How does it work ?

Decision tree is a type of **supervised learning algorithm** (having a pre-defined target variable) that is **mostly used in classification** problems.
It works for both **categorical and continuous input** and output variables.
In this technique, we split the population or sample into **two or more homogeneous sets** (or sub-populations) based on most significant splitter / differentiator in input variables.

Types of Decision Trees

1. **Categorical Variable Decision Tree**
2. **Continuous Variable Decision Tree: Has continuous target variable**

Important Terminology related to Decision Trees

1. **Root Node:**
2. **Splitting:**
3. **Decision Node:**
4. **Leaf/ Terminal Node:** Nodes do not split is called Leaf or Terminal node.
5. **Pruning:** When we remove sub-nodes of a decision node, this process is called pruning. You can say opposite process of splitting.
6. **Branch / Sub-Tree:**
7. **Parent and Child Node:**

Advantages:

1. **Easy to Understand:** Decision Tree looks like simple if-else statements which are very easy to understand and can be **visualized** easily.
2. **Useful in Data exploration:** one of the fastest way to identify most significant variables and relation between two or more variables.

3. **Less data cleaning required:** It is not influenced by **outliers** and **missing values** to a fair degree. No **feature scaling** required
4. **Data type is not a constraint:** It can handle both numerical and categorical variables.
5. **Can be used for both classification and regression problems.**
6. **Non Parametric Method:** Decision tree is considered to be a non-parametric method. This means that decision trees have **no assumptions** about the space distribution and the classifier structure.

Disadvantages

1. **Over fitting**. This problem gets solved by setting constraints on model parameters and pruning.
2. **High variance:** Due to the overfitting, there are very high chances of high variance in the output which leads to many errors in the final estimation and shows high inaccuracy in the results. In order to achieve zero bias (overfitting), it leads to high variance.
3. **Not fit for continuous variables**: While working with continuous numerical variables, decision tree looses information when it categorizes variables in different categories.
4. **Unstable**: Adding a new data point can lead to re-generation of the overall tree and all nodes need to be recalculated and recreated.
5. **Not suitable for large datasets**: If data size is large, then one single tree may grow complex and lead to overfitting. So in this case, we should use Random Forest instead of a single Decision Tree.

2. Regression Trees vs Classification Trees

1. Regression trees are used when **dependent variable** is continuous. Classification trees are used when dependent variable is categorical.
2. Regression tree: the value obtained by terminal nodes in the training data is the **mean** response of observation falling in that region.
   Classification tree: The value (class) obtained by terminal node in the training data is the **mode** of observations falling in that region..
3. Both the trees divide the predictor space (independent variables) into **distinct and non-overlapping regions**.
4. Both the trees follow a **top-down greedy approach** known as recursive binary splitting. ‘Greedy’ because, the algorithm cares (looks for best variable available) about only the current split, and not about future splits which will lead to a better tree.

5. This splitting process is continued until a user defined **stopping criteria** is reached.

3. How does a tree decide where to split / which feature to split on?

Decision trees use **multiple algorithms** to decide to split a node in two or more sub-nodes. The creation of sub-nodes increases the **homogeneity** of resultant sub-nodes. In other words, we can say that **purity** of the node increases with respect to the target variable. **Decision tree splits the nodes on all available variables and then selects the split which results in most homogeneous sub-nodes.**

The algorithm selection is also based on **type of target variables**. Let's look at the **four** most commonly used algorithms in decision tree:

1. Gini

Gini says, if we select two items from a population at random then they must be of same class and probability for this is 1 if population is pure.

1. It works with **categorical** target variable "Success" or "Failure".
2. It performs only Binary splits
3. **Higher the value of Gini higher the homogeneity.**
4. CART (Classification and Regression Tree) uses Gini method to create binary splits.

Steps to Calculate Gini for a split

1. Calculate Gini for sub-nodes, using the formula sum of square of probability for success and failure (p^2+q^2).
2. Calculate Gini for split using weighted Gini score of each node of that split

Also, Gini Impurity = 1-Gini

The Gini coefficient measures the inequality among values of a frequency distribution

2. Chi-Square

It is an algorithm to find out the **statistical significance between the differences between sub-nodes and parent node**. We measure it by sum of squares of standardized differences between observed and expected frequencies of target variable.

1. It works with **categorical** target variable "Success" or "Failure".

2. It can perform **two or more splits.**
3. **Higher the value of Chi-Square higher the statistical significance** of differences between sub-node and Parent node.
4. Chi-Square of each node is calculated using formula,
   Chi-square = ((Actual – Expected)^2 / Expected)^1/2

5. It generates tree called CHAID (Chi-square Automatic Interaction Detector)

Steps to Calculate Chi-square for a split:

1. Calculate Chi-square for individual node by calculating the deviation for Success and Failure both
2. Calculated Chi-square of Split using Sum of all Chi-square of success and Failure of each node of the split

Example, Split on Gender:

1. First we are populating for node Female, Populate the actual value for "**Play Cricket"** and **"Not Play Cricket"**, here these are 2 and 8 respectively.
2. Calculate expected value for "**Play Cricket"** and "**Not Play Cricket"**, here it would be 5 for both because parent node has probability of 50% and we have applied same probability on Female count(10).
3. Calculate deviations by using formula, Actual – Expected. It is for "**Play Cricket"** (2 – 5 = -3) and for "**Not play cricket"** (8 – 5 = 3).
4. Calculate Chi-square of node for "**Play Cricket**" and "**Not Play Cricket**" using formula with formula, **= ((Actual – Expected)^2 / Expected)^1/2**. You can refer below table for calculation.
5. Follow similar steps for calculating Chi-square value for Male node.
6. Now add all Chi-square values to calculate Chi-square for split Gender.

3. Information Gain:

Detailed Explanation: [Entropy: How Decision Trees Make Decisions | by Sam T | Towards Data Science]

Information theory is a measure to define this **degree of disorganization** in a system known as **Entropy**. If the sample is completely homogeneous, then the entropy is zero and if the sample is an equally divided (50% – 50%), it has entropy of one.

Entropy can be calculated using formula:- $\text{Entropy} = -p \log_2 p - q \log_2 q$

Here p and q are the probability of success and failure respectively in that node.

Entropy is also used with categorical target variables. It chooses the split which has lowest entropy compared to parent node and other splits. **The lesser the entropy, the better it is.**

Steps to calculate entropy for a split:

1. Calculate entropy of parent node
2. Calculate entropy of each individual node of split and calculate weighted average of all sub-nodes available in split.

Example: Let's use this method to identify the best split for student example.

1. Entropy for parent node = -(15/30) log2 (15/30) – (15/30) log2 (15/30) = **1**. Here 1 shows that it is a impure node.
2. Entropy for Female node = -(2/10) log2 (2/10) – (8/10) log2 (8/10) = 0.72 and for male node, -(13/20) log2 (13/20) – (7/20) log2 (7/20) = **0.93**
3. Entropy for split Gender = Weighted entropy of sub-nodes = (10/30)*0.72 + (20/30)*0.93 = **0.86**
4. Entropy for Class IX node, -(6/14) log2 (6/14) – (8/14) log2 (8/14) = 0.99 and for Class X node, -(9/16) log2 (9/16) – (7/16) log2 (7/16) = 0.99.
5. Entropy for split Class = (14/30)*0.99 + (16/30)*0.99 = **0.99**

Above, you can see that entropy for *Split on Gender* is the lowest among all, so the tree will split on *Gender*. We can derive information gain from entropy as **Entropy of Parent Class (=1 here) - Entropy of Splitted Nodes.**

$$IG(Y,X) = E(Y) - E(Y|X)$$

We simply subtract the entropy of Y given X from the entropy of just Y to calculate the reduction of uncertainty about Y given an additional piece of information X about Y. This is called Information Gain. **The greater the reduction in this uncertainty, the more information is gained about Y from X.**

Example:

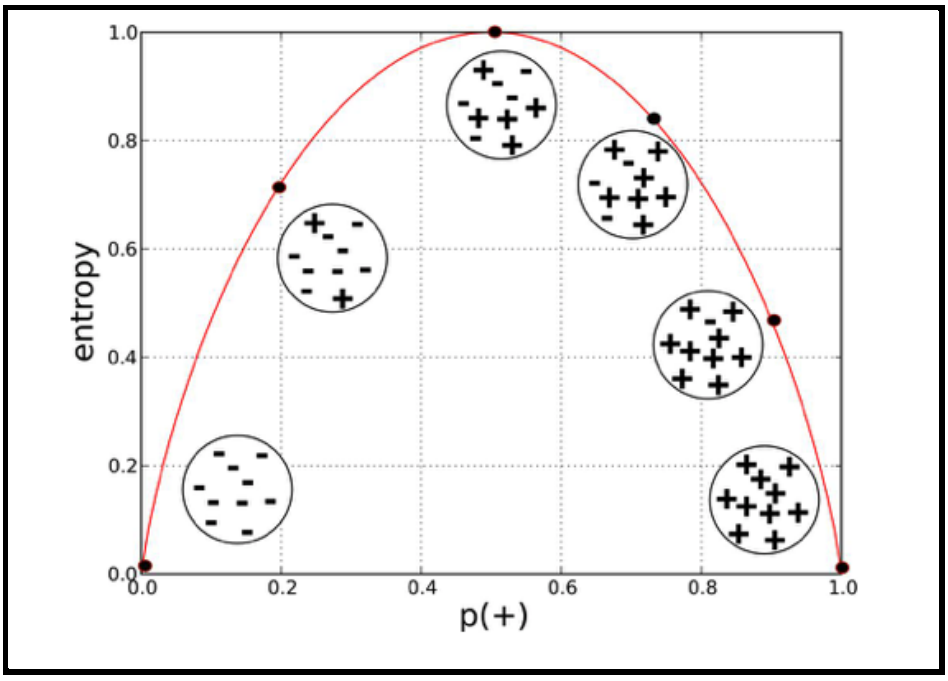
$$Information\ Gain:$$

$$IG(Liability, CR) = E(Liability) - E(Liability \mid CR)$$

$$= 1 - 0.625$$

$$= 0.375$$

CR = Credit Rating



Reduction in Variance

Reduction in variance is an algorithm used for **continuous target variables** (regression problems). This algorithm uses the standard formula of variance to choose the best split. The split with **lower variance is selected** as the criteria to split the population:

$$\text{Variance} = \frac{\Sigma(X - \overline{X})^2}{n}$$

Steps to calculate Variance:

1. Calculate variance for each node.
2. Calculate variance for each split as weighted average of each node variance.

4. When to stop splitting?

- If all features are exhausted -> no more feature left to split on
- If all the values are of 1 class i.e nodes are already homogeneous -> no more splits required to improve homogeneity
- Too few examples left in one node

5. What are the key parameters of tree modeling and how can we avoid over-fitting in decision trees?

Preventing overfitting is pivotal while modeling a decision tree and it can be done in 2 ways:

1. Setting constraints on tree size
2. Tree pruning

The parameters used for defining a tree are further explained below:

1.Minimum samples for a node split

- Defines the minimum number of samples (or observations) which are required in a node to be considered for splitting.
- Too high values can lead to under-fitting hence, it should be tuned using CV.

2. Minimum samples for a terminal node (leaf)

- Generally lower values should be chosen for imbalanced class problems because the regions in which the minority class will be in majority will be very small.

3. Maximum depth of tree (vertical depth)

- Should be tuned using CV.

4. Maximum number of terminal nodes

- Can be defined in place of max_depth. Since binary trees are created, a depth of ‘n’ would produce a maximum of 2^n leaves.

5. Maximum features to consider for split

- The number of features to consider while searching for a best split. These will be randomly selected.
- As a thumb-rule, **square root** of the total number of features works great but we should check upto **30-40%** of the total number of features.

- Higher values can lead to over-fitting but depends on case to case.

This is exactly the difference between normal decision tree & pruning. A decision tree with constraints won't see the truck ahead and adopt a greedy approach by taking a left. On the other hand if we use pruning, we in effect look at a few steps ahead and make a choice.

So we know pruning is better. But how to implement it in decision tree? The idea is simple.

1. We first make the decision tree to a large depth.
2. Then we start at the bottom and start removing leaves which are giving us negative returns when compared from the top.
3. Suppose a split is giving us a gain of say -10 (loss of 10) and then the next split on that gives us a gain of 20. A simple decision tree will stop at step 1 but in pruning, we will see that the overall gain is +10 and keep both leaves.

Note that sklearn's decision tree classifier does not currently support pruning. Advanced packages like xgboost have adopted tree pruning in their implementation. But the library *rpart* in R, provides a function to prune. Good for R users!

```
#Import Library
#Import other necessary libraries like pandas, numpy...
from sklearn import tree
#Assumed you have, X (predictor) and Y (target) for training data set and x_test(predictor) of test_dataset
# Create tree object
model = tree.DecisionTreeClassifier(criterion='gini') # for classification, here you can change the algorithm as gini or entropy (information gain) by default it is gini
# model = tree.DecisionTreeRegressor() for regression
# Train the model using the training sets and check score
model.fit(X, y)
model.score(X, y)
#Predict Output

predicted= model.predict(x_test)
```

7. What are ensemble methods in tree based modeling ?

Ensemble methods involve group of predictive models to achieve a better accuracy and model stability.
You build a small tree and you will get a model with low variance and high bias. . How do you manage to balance the trade off between bias and variance ?
Normally, as you increase the complexity of your model, you will see a reduction in prediction error due to lower bias in the model. As you continue to make your model more complex, you end up over-fitting your model and your model will start suffering from high variance.
Some of the commonly used ensemble methods include: Bagging, Boosting and Stacking.

8. What is Bagging? How does it work?

Bagging (bootstrap aggregation) is a technique used to **reduce the variance** of our predictions by **combining** the result of **multiple classifiers** modeled on **different sub-samples** of the same data set.

The steps followed in bagging are:

1. **Create Multiple DataSets:**
    - Sampling is done ***with replacement*** on the original data and new datasets are formed.
    - The new data sets can have a **fraction of the columns as well as rows**, which are generally hyper-parameters (a hyperparameter is a parameter whose value is set before the learning process begins. By contrast, the values of other parameters are derived via training) (max_features) in a bagging model
    - Taking row and column fractions less than 1 helps in making robust models, less prone to overfitting
2. **Build Multiple Classifiers:**
    - Classifiers are built on each data set.
    - Generally the same classifier is modeled on each data set and predictions are made.
3. **Combine Classifiers:**
    - The predictions of all the classifiers are combined using a mean, median or mode value depending on the problem at hand.
    - The combined values are generally more robust than a single model.

1. Multiple subsets are created from the original dataset, selecting observations with replacement.
2. A base model (weak model) is created on each of these subsets.
3. The models run in parallel and are independent of each other.
4. The final predictions are determined by combining the predictions from all the models.

There are various implementations of bagging models. Random forest is one of them.

This way **high variance of a single decision tree is reduced to low variance.**

9. What is Random Forest ? How does it work?

Random Forest is a versatile machine learning method capable of performing **both regression and classification tasks**. It also **undertakes dimensional reduction methods, treats missing values, outlier values** and other essential steps of data exploration, and does a fairly good job. It is a type of ensemble learning method, where a **group of weak models combine to form a powerful model.**

In Random Forest, we **grow multiple trees** as opposed to a single tree in CART mode
To classify a new object based on attributes, each tree gives a classification and we say the tree “votes” for that class. The forest chooses the classification having the **most votes** (over all the trees in the forest) and in case of regression, it takes the **average** of outputs by different trees
Tells us the **most important settings are the number of trees in the forest** (n_estimators) and the **number of features considered for splitting** at each leaf node (max_features).
It works in the following manner. Each tree is planted & grown as follows:

1. Assume number of cases in the training set is N. Then, sample of these **N cases is taken at random but *with replacement***. This sample will be the training set for growing the tree.

2. If there are **M input variables, a number m<M** is specified such that at each node, **m variables are selected at random** out of the M. The best split on these m is used to split the node. The value of m is held constant while we grow the forest.
3. Each tree is **grown to the largest extent possible** and there is **no pruning.**
4. **Predict** new data by **aggregating** the predictions of the ntree trees (i.e., majority votes for classification, average for regression).

Advantages of Random Forest

1. Solve both type of problems i.e. classification and regression
2. Power of handling large data sets with **higher dimensionality.** It can handle thousands of input variables and identify most significant variables.
3. It has an effective method for estimating **missing data** and maintains accuracy when a large proportion of the data are missing.
4. It has methods for balancing errors in data sets where **classes are imbalanced.**
5. Random Forest involves sampling of the input data **with replacement** called as **bootstrap sampling**. Here one third of the data is not used for training and can be used to testing. These are called the **out of bag** samples. Error estimated on these out of bag samples is known as *out of bag error*. Study of error estimates by Out of bag, gives evidence to show that the out-of-bag estimate is as accurate as using a test set of the same size as the training set. Therefore, using the out-of-bag error estimate removes the need for a set aside test set.
6. High variance of a single decision tree is reduced to low variance due to aggregation of multiple trees.

```
Import Library


from sklearn.ensemble import RandomForestClassifier #use RandomForestRegressor for regression problem


#Assumed you have, X (predictor) and Y (target) for training data set and x_test(predictor) of test_dataset


# Create Random Forest object
```

```
model= RandomForestClassifier(n_estimators=1000)

# Train the model using the training sets and check score

model.fit(X, y)

#Predict Output

predicted= model.predict(x_test)
```

10. What is Boosting ? How does it work?

The term 'Boosting' refers to a family of algorithms which converts w**eak learner to strong learners.**

'***How boosting identifies weak rules?***'

To find weak rule, we **apply base learning (ML) algorithms** with a different distribution. Each time base learning algorithm is applied, it generates a new weak prediction rule. This is an **iterative process**. After many iterations, the boosting algorithm combines these weak rules into a single strong prediction rule.

How do we choose different distribution for each round?'

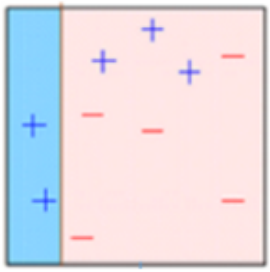
Step 1: *The base learner takes all the distributions and assign equal weight or attention to each observation.*
Step 2: *If there is any prediction error caused by first base learning algorithm, then* ***we pay higher attention to observations having prediction error****. Then, we apply the next base learning algorithm.*

Step 3: *Iterate Step 2 till the limit of base learning algorithm is reached or higher accuracy is achieved.*

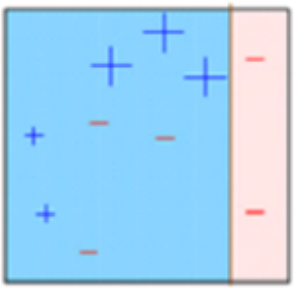
Boosting pays higher focus on examples which are mis-classified or have higher errors by preceding weak rules.

There are many boosting algorithms which ***impart additional boost to model's accuracy****. In this tutorial, we'll learn about the two most commonly used algorithms i.e.* ***Gradient Boosting*** *(GBM) and* ***XGboost****.*

1. *A subset is created from the original dataset.*
2. *Initially, all data points are given equal weights.*
3. *A base model is created on this subset.*
4. *This model is used to make predictions on the whole dataset.*



5. *Errors are calculated using the actual values and predicted values.*
6. *The observations which are incorrectly predicted, are given higher weights. (Here, the three misclassified blue-plus points will be given higher weights)*
7. *Another model is created and predictions are made on the dataset. (This model tries to correct the errors from the previous model)*



8. *Similarly, multiple models are created, each correcting the errors of the previous model.*
9. *The final model (strong learner) is the weighted mean of all the models (weak learners).*

Math Behind Gradient Boosting:

XGBoost Algorithm | XGBoost In Machine Learning (analyticsvidhya.com)

11. Which is more powerful: GBM or Xgboost?

Many advantages of Xgboost over GBM:

1. ***Regularization:*** *Standard GBM implementation has no regularization like XGBoost, therefore it also helps to reduce overfitting. In fact, XGBoost is also known as 'regularized boosting' technique.*
2. ***Parallel Processing:*** *XGBoost implements parallel processing and is* ***blazingly faster*** *as compared to GBM But hang on, we know that boosting is sequential process so* ***how can it be parallelized****? We know that each tree can be built only after the previous one, so what stops us from* ***making a tree using all cores****? XGBoost also* ***supports*** *implementation on* ***Hadoop****.*

3. ***High Flexibility:*** *XGBoost allow users to define* ***custom optimization objectives and evaluation criteria****. This adds a whole new dimension to the model and there is no limit to what we can do.*

4.Handling Missing Values: *XGBoost has an in-built routine to handle missing values.*

*User is required to supply a different value than other observations and pass that as a parameter. XGBoost tries different things as it encounters a missing value on each node and l****earns which path to take for missing values in future****.*

5. Tree Pruning: *A* ***GBM*** *would stop splitting a node when it encounters a negative loss in the split. Thus it is more of a* ***greedy algorithm.***

XGBoost *on the other hand make splits upto the max_depth specified and then start* ***pruning*** *the tree backwards and remove splits beyond which there is no positive gain.*

6. Built-in Cross-Validation

- *XGBoost allows user to run a cross-validation at each iteration of the boosting process and thus it is easy to* ***get the exact optimum number of boosting iterations*** *in a single run.*
- *This is unlike* ***GBM*** *where we have to* ***run a grid-search*** *and only a limited values can be tested.*

12. Working with GBM in R and Python

Overall pseudo-code of GBM algorithm for 2 classes:

1. Initialize the outcome

2. Iterate from 1 to total number of trees

2.1 Update the weights for targets based on previous run (higher for the ones mis-classified)

2.2 Fit the model on selected subsample of data

2.3 Make predictions on the full set of observations

2.4 Update the output with current results taking into account the learning rate

3. Return the final output.

Lets consider the important GBM parameters used to improve model performance in Python:

1. ***learning_rate***
    - *This determines the impact of each tree on the final outcome (step 2.4). GBM works by starting with an initial estimate which is updated using the output of each tree. The learning parameter* ***controls the magnitude of this change in the estimates.***
    - ***Lower values*** *are generally preferred as they make the model* ***robust*** *to the specific characteristics of a tree and thus allowing it to generalize well.*
    - *Lower values would require higher number of trees to model all the relations and will be* ***computationally expensive.***
2. ***n_estimators***
    - *The number of sequential trees to be modeled (step 2)*

- *Though GBM is fairly robust at higher number of trees but it can still overfit at a point. Hence, this should be **tuned using CV for a particular learning rate.***

3. ***Subsample***
   *The **fraction of observations to be selected for each tree**. Selection is done by random sampling.*
   Values slightly less than 1 make the model robust by reducing the variance.
   ***Typical values ~0.8** generally work fine but can be fine-tuned further.*

4. loss:*It refers to the loss function to be minimized in each split.*

5. ***Init:*** *This affects initialization of the output. This can be **used if we have made another model whose outcome is to be used as the initial estimates for GBM**.*

6. ***Random_state:*** *The random number seed so that same random numbers are generated every time. This is important for parameter tuning. If we don't fix the random number, then we'll have different outcomes for subsequent runs on the same parameters and it becomes **difficult to compare models.***

7. Verbose: *The type of output to be printed when the model fits. The different values can be:*

   - *0: no output generated (default)*
   - *1: output generated for trees in certain intervals*
   - *>1: output generated for all trees*

2. warm_start
   - *This parameter has an interesting application and can help a lot if used judicially.*
   - *Using this, we can fit additional trees on previous fits of a model. It can save a lot of time and you should explore this option for advanced applications*

3. presort
   - *Select whether to presort data for faster splits.*
   - *It makes the selection automatically by default but it can be changed if needed.*

GBM in Python

```
#import libraries

from sklearn.ensemble import GradientBoostingClassifier #For Classification

from sklearn.ensemble import GradientBoostingRegressor #For Regression #use GBM function

clf = GradientBoostingClassifier(n_estimators=100, learning_rate=1.0, max_depth=1)

clf.fit(X_train, y_train)
```

13. Working with XGBoost in R and Python

It's feature to implement parallel computing makes it at least 10 times faster than existing gradient boosting implementations. It supports various objective functions, including regression, classification and ranking.

Complete Guide to Parameter Tuning in Gradient Boosting (GBM) in Python

2. GBM Parameters

The overall parameters can be divided into 3 categories:

1. ***Tree-Specific Parameters:*** *These affect each individual tree in the model.*
2. ***Boosting Parameters:*** *These affect the boosting operation in the model.*
3. ***Miscellaneous Parameters:*** *Other parameters for overall functioning.*

1.Tree specific parameters

A. ***min_samples_split***

Defines the minimum number of samples (or observations) which are required in a node to be considered for splitting. Too high values can lead to under-fitting hence, it should be tuned using CV.

B. min_samples_leaf:

Defines the minimum samples (or observations) required in a terminal node or leaf.

Generally lower values should be chosen for imbalanced class problems because the regions in which the minority class will be in majority will be very small.

C. min_weight_fraction_leaf:

Similar to min_samples_leaf but defined as a fraction of the total number of observations instead of an integer.

Only one of #2 and #3 should be defined.

***D. max_depth:** Should be tuned using CV.*

E. max_leaf_nodes
The maximum number of terminal nodes or leaves in a tree.

*Can be defined **in place of max_depth.** Since binary trees are created, a depth of 'n' would produce a maximum of 2^n leaves.*

If this is defined, GBM will ignore max_depth.

F. max_features

The number of features to consider while searching for a best split. These will be randomly selected.

*As a thumb-rule, s**quare root of the total number of features** works great but we should **check upto 30-40% of the total number of features**.*

Higher values can lead to over-fitting but depends on case to case.

2. Boosting parameters:

A. ***Learning_rate:*** *This determines the impact of each tree on the final outcome (step 2.4). GBM works by starting with an initial estimate which is updated using the output of each tree. The learning parameter controls the magnitude of this change in the estimates.*
   Lower values are generally preferred as they make the model robust to the specific characteristics of tree and thus allowing it to generalize well.
   Lower values would require higher number of trees to model all the relations and will be computationally expensive.
B. ***n_estimators:*** ***T****he number of sequential trees to be modeled. Should be tuned using CV for a particular learning rate.*
C. ***Subsample***
   The fraction of observations to be selected for each tree. Selection is done by random sampling.
   Values slightly less than 1 make the model robust by reducing the variance.
   Typical values ~0.8 generally work fine but can be fine-tuned further.

3. Miscellaneous parameters:

loss

- *It refers to the loss function to be minimized in each split.*
- *It can have various values for classification and regression case. Generally the default values work fine. Other values should be chosen only if you understand their impact on the model.*

init

- *This affects initialization of the output.*
- *This can be used if we have made another model whose outcome is to be used as the initial estimates for GBM.*

random_state

- *The random number seed so that same random numbers are generated every time.*
- *This is important for parameter tuning. If we don't fix the random number, then we'll have different outcomes for subsequent runs on the same parameters and it becomes difficult to compare models.*

- *It can potentially result in overfitting to a particular random sample selected. We can try running models for different random samples, which is computationally expensive and generally not used.*

verbose

- *The type of output to be printed when the model fits. The different values can be:*
    - *0: no output generated (default)*
    - *1: output generated for trees in certain intervals*
    - *>1: output generated for all trees*

warm_start

- *This parameter has an interesting application and can help a lot if used judicially.*
- *Using this, we can fit additional trees on previous fits of a model. It can save a lot of time and you should explore this option for advanced applications*

presort

- *Select whether to presort data for faster splits.*
- *It makes the selection automatically by default but it can be changed if needed.*

```
#Import libraries:

import pandas as pd

import numpy as np

from sklearn.ensemble import GradientBoostingClassifier  #GBM algorithm

from sklearn import cross_validation, metrics   #Additional scklearn functions

from sklearn.grid_search import GridSearchCV   #Perforing grid search


import matplotlib.pylab as plt

%matplotlib inline

from matplotlib.pylab import rcParams
```

```
rcParams['figure.figsize'] = 12, 4

train = pd.read_csv('train_modified.csv')

target = 'Disbursed'

IDcol = 'ID'

def modelfit(alg, dtrain, predictors, performCV=True, printFeatureImportance=True, cv_folds=5):

    #Fit the algorithm on the data

    alg.fit(dtrain[predictors], dtrain['Disbursed'])



    #Predict training set:

    dtrain_predictions = alg.predict(dtrain[predictors])

    dtrain_predprob = alg.predict_proba(dtrain[predictors])[:,1]



    #Perform cross-validation:

    if performCV:

        cv_score = cross_validation.cross_val_score(alg, dtrain[predictors], dtrain['Disbursed'], cv=cv_folds, scoring='roc_auc')



    #Print model report:

    print "\nModel Report"

  print "Accuracy : %.4g" % metrics.accuracy_score(dtrain['Disbursed'].values, dtrain_predictions)
```

```
    print "AUC Score (Train): %f" % metrics.roc_auc_score(dtrain['Disbursed'], dtrain_predprob)


    if performCV:

        print "CV Score : Mean - %.7g | Std - %.7g | Min - %.7g | Max - %.7g" % (np.mean(cv_score),np.std(cv_score),np.min(cv_score),np.max(cv_score))


    #Print Feature Importance:

    if printFeatureImportance:

        feat_imp = pd.Series(alg.feature_importances_, predictors).sort_values(ascending=False)

        feat_imp.plot(kind='bar', title='Feature Importances')

        plt.ylabel('Feature Importance Score')
```

General Approach for Parameter Tuning

1. *Choose a relatively **high learning rate**. Generally the default value of 0.1 works but somewhere between **0.05 to 0.2** should work for different problems*
2. *Determine the **optimum number of trees** for this learning rate. This should range around **40-70**. Remember to choose a value on which your system can work fairly fast. This is because it will be used for testing various scenarios and determining the tree parameters.*
3. *Tune **tree-specific parameters** for decided learning rate and number of trees. Note that we can choose different parameters to define a tree and I'll take up an example here.*
4. ***Lower the learning rate** and increase the estimators proportionally to get more robust models.*

Fix learning rate and number of estimators for tuning tree-based parameters

1. ***min_samples_split = 500*** *: This should be ~0.5-1% of total values. Since this is imbalanced class problem, we'll take a small value from the range.*
2. ***min_samples_leaf = 50*** *: Can be selected based on intuition. This is just used for preventing overfitting and again a small value because of imbalanced classes.*
3. ***max_depth = 8*** *: Should be chosen (5-8) based on the number of observations and predictors. This has 87K rows and 49 columns so lets take 8 here.*
4. ***max_features = 'sqrt'*** *: Its a general thumb-rule to start with square root.*
5. *subsample = 0.8 : This is a commonly used used start value*

Tuning tree-specific parameters

Now lets move onto tuning the tree parameters. I plan to do this in following stages:

1. *Tune max_depth and num_samples_split*
2. *Tune min_samples_leaf*
3. *Tune max_features*

The order of tuning variables should be decided carefully. You should take the variables with a higher impact on outcome first. For instance, max_depth and min_samples_split have a significant impact and we're tuning those first.

Complete Guide to Parameter Tuning in XGBoost (with codes in Python)

1. ***General Parameters:*** *Guide the overall functioning*
2. ***Booster Parameters:*** *Guide the individual booster (tree/regression) at each step*
3. ***Learning Task Parameters:*** *Guide the optimization performed*

Booster Parameters

*Though there are 2 types of boosters, I'll consider only a tree **booster** here because it always outperforms the linear booster and thus the latter is rarely used.*

1. ***eta [default=0.3]***
    - *Analogous to learning rate in GBM*
    - *Makes the model more robust by shrinking the weights on each step*
    - *Typical final values to be used: 0.01-0.2*
2. ***min_child_weight [default=1]***
    - *Here, min_child_weight means something like **"stop trying to split once you reach a certain degree of purity in a node and your model can fit it"**.*
    - *Defines the minimum sum of weights of all observations required in a child.*
    - *This is similar to **min_child_leaf** in GBM but not exactly. This refers to min "sum of weights" of observations while GBM has min "number of observations".*
    - *Used to control over-fitting. Higher values prevent a model from learning relations which might be highly specific to the particular sample selected for a tree.*
    - *Too high values can lead to under-fitting hence, it should be tuned using CV.*
3. ***max_depth [default=6]***
    - *The maximum depth of a tree, same as GBM.*
    - *Used to control over-fitting as higher depth will allow model to learn relations very specific to a particular sample.*
    - *Should be tuned using CV.*
    - *Typical values: 3-10*
4. ***max_leaf_nodes***
    - *The maximum number of terminal nodes or leaves in a tree.*
    - *Can be defined in place of max_depth. Since binary trees are created, a depth of 'n' would produce a maximum of 2^n leaves.*
    - *If this is defined, GBM will ignore max_depth.*
5. ***gamma [default=0]***
    - *A node is split only when the resulting split gives a positive reduction in the loss function. Gamma specifies the minimum loss reduction required to make a split.*
    - *Makes the algorithm conservative. The values can vary depending on the loss function and should be tuned.*
6. ***max_delta_step [default=0]***
    - *In maximum delta step we allow each tree's weight estimation to be. If the value is set to 0, it means there is no constraint. If it is set to a positive value, it can help making the update step more conservative.*

- *Usually this parameter is not needed, but it might help in logistic regression when class is extremely imbalanced.*
- *This is generally not used but you can explore further if you wish.*

7. ***subsample [default=1]***
   - *Same as the subsample of GBM. Denotes the fraction of observations to be randomly samples for each tree.*
   - *Lower values make the algorithm more conservative and prevents overfitting but too small values might lead to under-fitting.*
   - *Typical values: 0.5-1*
8. ***colsample_bytree [default=1]***
   - *Similar to max_features in GBM. Denotes the fraction of columns to be randomly samples for each tree.*
   - *Typical values: 0.5-1*
9. ***colsample_bylevel [default=1]***
   - *Denotes the subsample ratio of columns for each split, in each level.*
   - *I don't use this often because subsample and colsample_bytree will do the job for you. but you can explore further if you feel so.*
10. ***lambda [default=1]***
    - *L2 regularization term on weights (analogous to Ridge regression)*
    - *This used to handle the regularization part of XGBoost. Though many data scientists don't use it often, it should be explored to reduce overfitting.*
11. ***alpha [default=0]***
    - *L1 regularization term on weight (analogous to Lasso regression)*
    - *Can be used in case of very high dimensionality so that the algorithm runs faster when implemented*
12. ***scale_pos_weight [default=1]***
    - *A value greater than 0 should be used in case of high class imbalance as it helps in faster convergence.*

Learning Task Parameters

These parameters are used to define the optimization objective the metric to be calculated at each step.

1. ***objective [default=reg:linear]***
   - *This defines the loss function to be minimized. Mostly used values are:*
     - ***binary:logistic** –logistic regression for binary classification, returns predicted probability (not class)*

        - ***multi:softmax*** *–multiclass classification using the softmax objective, returns predicted class (not probabilities)*
            - *you also need to set an additional* ***num_class*** *(number of classes) parameter defining the number of unique classes*
        - ***multi:softprob*** *–same as softmax, but returns predicted probability of each data point belonging to each class.*

eval_metric [default according to objective]

- *The metric to be used for validation data.*
- *The default values are rmse for regression and error for classification.*
- *Typical values are:*
    - ***rmse*** *– root mean square error*
    - ***mae*** *– mean absolute error*
    - ***logloss*** *– negative log-likelihood*
    - ***error*** *– Binary classification error rate (0.5 threshold)*
    - ***merror*** *– Multiclass classification error rate*
    - ***mlogloss*** *– Multiclass logloss*
    - ***auc:*** *Area under the curve*

seed [default=0]

- *The random number seed.*
- *Can be used for generating reproducible results and also for parameter tuning.*

You must be wondering that we have defined everything except something similar to the ***"n_estimators" parameter in GBM****. Well this exists as a parameter in* ***XGBClassifier****. However, it has to be passed as* ***"num_boosting_rounds"*** *while calling the fit function in the standard xgboost implementation.*

Find out more about the following:

1. **Bootstrap sample**: The idea behind bagging is combining the results of multiple models (for instance, all decision trees) to get a generalized result. Here's a question: If you create all the models on the same set of data and combine it, will it be useful? There is a high chance that these models will give the same result since they are getting the same input. So how can we solve this problem? One of the techniques is bootstrapping.

2. Bootstrapping is a sampling technique in which we create subsets of observations from the original dataset, with replacement. The size of the subsets is the same as the size of the original set.

3. **Max_features parameter:** to determine the no of features in the subset of features from which the best split is chosen
   If max_features = 1 very different trees
   If max_features <= close to total features, similar trees

4. **n_jobs**: how many cores running in parallel during training
   (if n_jobs= -1, then uses all cores)

Other Boosting algorithms:
(https://www.analyticsvidhya.com/blog/2018/06/comprehensive-guide-for-ensemble-models/
)

Light GBM beats all the other algorithms when the dataset is extremely large. Compared to the other algorithms, Light GBM takes lesser time to run on a huge dataset. LightGBM is a gradient boosting framework that uses tree-based algorithms and **follows leaf-wise approach** while other algorithms work in a level-wise approach pattern. The images below will help you understand the difference in a better way.

CatBoost: Handling categorical variables is a tedious process, especially when you have a large number of such variables. When your categorical variables have too many labels (i.e. they are highly cardinal), performing one-hot-encoding on them exponentially increases the dimensionality and it becomes really difficult to work with the dataset.
CatBoost can automatically deal with categorical variables and does not require extensive data preprocessing like other machine learning algorithms

AdaBoost

One way for a new predictor to correct its predecessor is to **pay a bit more attention to the training instances that the predecessor underfitted**. This results in new predictors focusing more and more on the hard cases. This is the technique used by AdaBoost. For example, to build an AdaBoost classifier, a first base classifier (such as a Decision Tree) is trained and used to make predictions on the training set. The relative weight of misclassified

training instances is then increased. A second classifier is trained using the updated weights and again it makes predictions on the training set, weights are updated, and so on

Another very popular Boosting algorithm is Gradient Boosting.17 Just like AdaBoost, Gradient Boosting works by sequentially adding predictors to an ensemble, each one correcting its predecessor. However, instead of tweaking the instance weights at every iteration like AdaBoost does, this method tries to fit the new predictor to the residual errors made by the previous predictor.

Stacking

The last Ensemble method we will discuss in this chapter is called stacking (short for stacked generalization).18 It is based on a simple idea: instead of using trivial functions (such as hard voting) to aggregate the predictions of all predictors in an ensemble, why don't we train a model to perform this aggregation? Figure 7-12 shows such an ensemble performing a regression task on a new instance. Each of the bottom three predictors predicts a different value (3.1, 2.7, and 2.9), and then the final predictor (called a blender, or a meta learner) takes these predictions as inputs and makes the final prediction (3.0).

LIGHTGBM

Q)Differentiate between Bagging and Boosting.

Ans): Boosting and bagging are types of ensemble learning.

Ensemble learning is a method that is used to enhance the performance of ML algo by combining several learners. When compared to a single model, this type of learning builds models with improved efficiency and accuracy

Boosting: **Sequential** ensemble - Weak learners are sequentially produced during the training. The performance of the model is improved by assigning higher weightage to incorrectly classified samples. Feed entire data-> makes prediction-> misclassified some example-> more weightage to these misclassified examples.

Bagging (Bootstrap Aggregation): Parallel ensemble - Performance of model improved by **parallely** training a no of weak learners on a Bootstrapped dataset.

Q)What is Gradient Boosting? How is it different from normal boosting?

Ans: Base learners are sequentially generated such that present base learner is always more effective than previous base learner

Here the weights for misclassified outcomes are not incremented.

Instead we try to minimize the loss function of the previous learner.

Eg: XGBoost - Advanced grad boosting - Xtreme Grad Boosting - much faster

Supports parallelization, implements distributed computing model, cache optimization, out of core optimization

Exercises

1. If you have trained five different models on the exact same training data, and they all achieve 95% precision, is there any chance that you can combine these models to get better results? If so, how? If not, why?
2. What is the difference between hard and soft voting classifiers?
3. Is it possible to speed up training of a bagging ensemble by distributing it across multiple servers? What about pasting ensembles, boosting ensembles, random forests, or stacking ensembles?
4. What is the benefit of out-of-bag evaluation?
5. What makes Extra-Trees more random than regular Random Forests? How can this extra randomness help? Are Extra-Trees slower or faster than regular Random Forests?
6. If your AdaBoost ensemble underfits the training data, what hyperparameters should you tweak and how?
7. If your Gradient Boosting ensemble overfits the training set, should you increase or decrease the learning rate?

1. If you have trained five different models and they all achieve 95% precision, you can try combining them into a voting ensemble, which will often give you even better results. It works better if the models are very different (e.g., an SVM classi- fier, a Decision Tree classifier, a Logistic Regression classifier, and so on). It is even better if they are trained on different training instances (that's the whole point of bagging and pasting ensembles), but if not it will still work as long as the models are very different.

2. A hard voting classifier just counts the votes of each classifier in the ensemble and picks the class that gets the most votes. A soft voting classifier computes the average

estimated class probability for each class and picks the class with the highest probability. This gives high-confidence votes more weight and often performs better, but it works only if every classifier is able to estimate class probabilities (e.g., for the SVM classifiers in Scikit-Learn you must set probability=True).

3. It is quite possible to speed up training of a bagging ensemble by distributing it across multiple servers, since each predictor in the ensemble is independent of the others. The same goes for pasting ensembles and Random Forests, for the same reason. However, each predictor in a boosting ensemble is built based on the previous predictor, so training is necessarily sequential, and you will not gain anything by distributing training across multiple servers. Regarding stacking ensembles, all the predictors in a given layer are independent of each other, so they can be trained in parallel on multiple servers. However, the predictors in one layer can only be trained after the predictors in the previous layer have all been trained.

4. With out-of-bag evaluation, each predictor in a bagging ensemble is evaluated using instances that it was not trained on (they were held out). This makes it pos- sible to have a fairly unbiased evaluation of the ensemble without the need for an additional validation set. Thus, you have more instances available for training, and your ensemble can perform slightly better.

5. When you are growing a tree in a Random Forest, only a random subset of the features is considered for splitting at each node. This is true as well for ExtraTrees, but they go one step further: rather than searching for the best possible thresholds, like regular Decision Trees do, they use random thresholds for each feature. This extra randomness acts like a form of regularization: if a Random Forest overfits the training data, Extra-Trees might perform better. Moreover, since Extra-Trees don’t search for the best possible thresholds, they are much faster to train than Random Forests. However, they are neither faster nor slower than Random Forests when making predictions.

6. If your AdaBoost ensemble underfits the training data, you can try increasing the number of estimators or reducing the regularization hyperparameters of the base estimator. You may also try slightly increasing the learning rate.

7. If your Gradient Boosting ensemble overfits the training set, you should try decreasing the learning rate. You could also use early stopping to find the right number of predictors (you probably have too many).

Q.What is pruning and what are the different types of pruning ?

Q.For a specific dataset, it is observed that a linear regression model outperforms all tree based models. What conclusions can be drawn about the dataset from this ?

Q.Why are decision trees prone to overfitting ? How can this issue be addressed?

Interview-Prepartion-Data-Science/Interview Preparation- Day 4- Decision Trees.ipynb at master · krishnaik06/Interview-Prepartion-Data-Science (github.com)