Table of Contents

....................3
Introduction....................3
Statistical Learning....................7
Probably Approximately Correct (PAC Method)....................9
Linear Regression....................11
Simple Linear Regression....................11
Multiple Linear Regression....................17
Simple Linear Algorithm....................18
Gradient descent....................23
Matrix....................25
Multiple Linear Algorithm....................28
Pandas....................31
Model file handling....................38
Model Creation....................38
Prediction....................42
Flask API....................43
Model comparison....................45
Improving Model score....................48
Feature Selection....................53
Highly Correlated features (MV - DV)....................53
Zero Correlated features (PV - MV,DV)....................56
Recursive Feature Elimination Technique....................57
Outliers Removal....................58
Explanatory Data Analysis....................61
Univariate....................61
Bivariate....................65
Multivariate....................69
Polynomial Regression....................72
Underfitting – High bias....................75
Overfitting – High variance....................75
Regularization....................76
Logistic regression....................77
Sigmoid function....................78
Decision Boundary....................78
Cost function....................79
Classification accuracy....................81
Confusion Matrix....................81
Precision, Recall & F1 score....................82
Trading-off between Precision & Recall....................83
Multi-class classification....................83
Vectors....................87
Natural Language Toolkit....................90
Decision Trees & Random Forest....................93
Clustering with K-Means....................99
Centroids (திணிவுக்கான புள்ளி)....................101
Elbow Method....................104
silhouette_coefficient....................106
SVM....................108

Large margin classifier (linear)....109
Kernels (non-linear)....111
PCA....113
Data Projection....116
Projection Error....116
Compressed components....117
Neural Networks....118
Neural Network அமைப்பு....119
h(x) கணிப்புகள் நிகழும் விதம்....120
Forward propagation....123
Back propagation....124
Perceptron....124
Artificial Neural Networks....129

ஆசிரியர் உரை:

அனைவரையும் போல எனக்கும் ஒருநாள் வேலைப்பார்க்கும் இடத்தில் பல்வேறு பிரச்சனைகள் எழத் தொடங்கியது. அலுவலகத்தில் அதிக நேரம் இல்லாதது, இருக்கும் நேரத்திலும் உடற்பயிற்சிக் கூடம் செல்வது என்று பல்வேறு புகார்கள் என்மீது வைக்கப்பட்டன. பன்னாட்டு நிறுவனங்களில் சுதந்திரமாக இருந்து வேலை பார்த்துப் பழகிய எனக்கு, திடீரென்று ஒரு சிறிய நிறுவனத்தில் வேலை பார்க்கத் தொடங்கியதால் ஏற்பட்ட பிரச்சனையே இது. இத்தகைய பணிச்சூழல் எனக்கு சற்றும் ஒத்துப் போகவில்லை. எனவே மீண்டும் பன்னாட்டு நிறுவனத்திற்கே சென்று விடலாம் என்ற யோசனையும் இல்லாமல், சட்டென்று ஒரு நாள் வேலையை விட்டுவிட்டேன். நமக்குத்தான் 10 வருடம் 'Software Testing' துறையில் அனுபவம் உள்ளதே எனும் நினைப்பில், புது வேலை கிடைக்கும் முன்பே இருக்கும் வேலையை விட்டுவிட்டேன். ஆனால் என்னுடைய அந்த 10 வருட அனுபவமே எனக்கு மிகப் பெரிய பிரச்சனையாக அமைந்துவிட்டது. எங்கு நேர்காணலுக்குச் சென்றாலும், இவ்வளவு அதிக அனுபவம் கொண்ட நபர் எங்களுக்கு Testing-க்குத் தேவையில்லை என்று கூறி அனுப்பிவிட்டனர். வீட்டில் இருந்தே பழக்கப்படாத எனக்கு, வேலையில்லாமல் வீட்டில் இருப்பது சற்று கடினமான காலகட்டமாகவே இருந்தது. எங்கு சென்றாலும் தொடர்ச்சியாக நிராகரிக்கப் பட்டேன். Testing செய்ய குறைந்த அனுபவம் கொண்டவர்களே போதும் என்றார்கள். எனவே Testing என்று சொன்னால், என்னுடைய அனுபவத்திற்கு வேலை கிடைப்பது கடினம் என்று தோன்றவே, வேறு துறைகளில் என்னுடைய திறன்களை வளர்த்துக் கொள்ள முடிவு செய்தேன். இதுவரை தொடர்ச்சியாக நான் கற்றுக்கொண்டே வந்த Linux, Mysql, html, css, javascript போன்றவை எனக்கு கைகொடுக்கத் தொடங்கின. அவைகளோடு சேர்த்து ELK, hadoop, pig, hive, spark போன்றவற்றையும் கற்கத் தொடங்கினேன். அதற்காகத் தனியாக எந்த ஒரு course-ம் சேரவில்லை. course நடத்துபவர்கள் அதற்கான பயிற்சித் தொகையாக யானை விலை, குதிரை விலை கேட்டார்கள். எனவே வீட்டில் இருந்தபடியே கற்கத் தொடங்கினேன். அதன் பின்னர் Bigdata Engineer, Hadoop admin போன்ற பதவிக்கான நேர்காணல்களை எதிர்கொள்ளத் தொடங்கினேன். இவை எல்லாம் development skills-ன் கீழ் வராது. தனியாக digital skills என்பதன் கீழ் வரும். நேர்காணலில் கேள்விகள் கேட்போருக்கும் நமக்குத் தெரிந்த அளவே தெரிந்திருக்கும். எனவே சுலபமாக தேர்வாகிவிடலாம். இதில் ஆச்சரியம் என்னவென்றால், நான் பங்கு பெற்ற இரண்டு நிறுவனங்களின் நேர்காணலிலும் என்னைத் தேர்வு செய்து விட்டார்கள். அதில் ஒன்றான TCS -ல் நான் வேலைக்குச் சேர்ந்துவிட்டேன். அதன் பின்னர் வழக்கம்போல எனக்குத் தெரிந்த bigdata, hadoop போன்றவற்றை விட்டுவிட்டு, 'Machine learning' எனும் புதிய துறையில் வேலைசெய்யச் சொன்னார்கள். நானும் ஆர்வமுடன் பல புதிய விஷயங்களைக் கற்றுக் கொண்டு சுதந்திரமாக வேலை செய்யத் தொடங்கினேன். இதுவரை நான் கற்றுக்கொண்ட விஷயங்களை வைத்து இப்புத்தகத்தை எழுதியுள்ளேன். தற்போது Deep learning-ஐப் பற்றிக் கற்றுக் கொண்டு இருக்கிறேன். இதனால் நான் சொல்ல வருவது என்னவென்றால், "கற்றோருக்கு செல்லும் இடமெல்லாம் சிறப்பு" :)

Introduction

இயந்திரவழிக் கற்றல் என்பது தற்போது அதிகமாக வளர்ந்து வருகின்ற ஒரு துறை. ஒரு கணினிக்கு கற்பிப்பது, அதற்கு அறிவு புகட்டுவது, புகட்டப்பட்ட அறிவின் அடிப்படையில் கணினிகளையே முடிவினை மேற்கொள்ளுமாறு செய்வது போன்ற பல்வேறு விஷயங்களை இயந்திரவழிக் கற்றலில் காணலாம். மனிதன் செய்கின்ற வேலையை வெறும் நிரல்கள் எழுதி கணினியைச் செய்யவைப்பதன் பெயர் இயந்திரவழிக் கற்றல் ஆகாது. அதன் பெயர் தானியக்கம் (Automation). மனிதனைப் போன்று கணினிகளை யோசிக்க வைத்து, முடிவுகளையும் அதனை வைத்தே எடுக்க வைப்பது, அவ்வாறு எடுக்கப்படும் முடிவுகள் இயந்திரத்தனமாக அல்லாமல் அறிவின் அடிப்படையில் அமைவதற்கு என்னென்ன செய்ய வேண்டும், அவ்வாறு யோசிக்க வைப்பது எவ்வாறு சாத்தியப்பட்டது, அதிலுள்ள வழிமுறைகள் என்ன, கோட்பாடுகள் என்னென்ன என்பது போன்ற அனைத்தையும் விளக்குவதே இயந்திரவழிக் கற்றல் ஆகும். இவற்றையெல்லாம் செய்வதற்கு வெறும் தகவல் தொழில்நுட்ப அறிவோடு மட்டுமல்லாமல், கணிதம், புள்ளியியல் போன்ற மற்ற துறைகளிலும் சிறிது அடிப்படை அறிவினை வளர்த்துக் கொள்ள வேண்டும். அப்போதுதான் நம்மால் சுலபமாக கணினிக்குக் கற்றுக் கொடுக்க முடியும். மேலும் மாறும் சூழ்நிலைகளுக்கும், தரவுகளுக்கும் ஏற்ப தமது சோதனை முடிவுகளையும், கணிப்புகளையும் மாற்றி வழங்குவதே இயந்திரவழிக் கற்றலின் சிறப்புப் பண்பு ஆகும். இதையே Adaptivity என்று கூறுவர். எந்தெந்த வகையான சூழ்நிலைகள் இயந்திரவழிக் கற்றலுக்கு வழி வகுக்கின்றன என்பதைக் கீழே காணலாம்.

<u>மனிதனுடைய அனுபவ அறிவால் செய்யக்கூடிய செயல்கள்:</u> *வாகனம் ஓட்டுவது, ஒருவருடைய குரலைக் கேட்டே ஆளை கணிப்பது போன்றவையெல்லாம் ஒருவர் தன்னுடைய அனுபவ அறிவாலும், புத்தி சாதுர்யத்தாலும் செய்யக் கூடியவை. இவற்றுக்கெல்லாம் நேரடியாக நிரல்கள் எழுதி கணினிக்கு சொல்லித்தர முடியாது. அந்த அனுபவத்தையும் அறிவையும் கொடுத்துத்தான் நாம் கணினியைப் பழக்க வேண்டும். மேலும் எத்துறை சார்ந்த விஷயங்களை நாம் கணினிக்குப் புகட்ட விரும்புகிறோமோ, அத்துறை சார்ந்த வல்லுநர்கள் மூலம் கணினிக்குப் போதிய அறிவினை வழங்க வேண்டும். இதையே domain expertise என்று கூறுவர்.*

<u>மனித சக்தியை மீறி செய்ய வேண்டிய செயல்கள்:</u> *விண்வெளி, பூகோளம், அறிவியல் போன்ற அனைத்துத் துறைகளிலும் பல்வேறு சோதனைகள் நடத்தப்படுகின்றன. அவை வெற்றியைத் தழுவ வேண்டுமென்றால், ஏற்கெனவே தோல்வியைத் தழுவிய முந்தைய சோதனை முடிவுகளை எடுத்து ஆராய்ந்து வெற்றிக்கான கணிப்பினைக் கூற வேண்டும். உதாரணத்துக்கு மருத்துவத் துறையை எடுத்துக்கொண்டால், பிரசவத்தின்போது பெண்களின் இறப்பு விகிதம் என்பது பாதிக்குப் பாதியாக இருந்தது. இதனைக் குறைப்பதற்கு இதுவரை பிரசவத்தின்போது இறந்த பலகோடிக்கணக்கான பெண்களின் ஆய்வறிக்கைகளை எடுக்க வேண்டும். அவற்றுள் ஒவ்வொரு பெண்ணும் எதனால் இறந்திருக்கிறாள், எத்தனை பெண்கள் ஒரே வகையான காரணத்தால் இறக்கிறார்கள், எத்தனை வகையான காரணங்கள் இறப்புக்கு வழிவகுக்கின்றன என்பது போன்ற விஷயங்களையெல்லாம் கண்டு பிடிக்க வேண்டும். இதெல்லாம் உண்மையிலேயே மனித சக்திக்கு அப்பாற்பட்ட செயல்தான். ஆகவே இவற்றைச் செய்வதற்கு கணினிகளைப் பழக்கி, ஒரே மாதிரியான pattern-ல் இருக்கும் தரவுகளைக் கண்டுபிடிக்கின்றனர். பின்னர் அவற்றை எடுத்து மருத்துவ வல்லுநர்கள் பரிசீலித்து "இறப்புக்கு வழிவகுக்கும் காரணிகள்" என முடிவு செய்கின்றனர். பின்னர் இதன்*

அடிப்படையில்தான் தற்போது கர்பிணியாக வருகின்ற பெண்களிடம் இவற்றில் ஏதேனும் ஒன்று தென்பட்டால் கூட உடனே அறுவை சிகிச்சை செய்து விடுகின்றனர். ஆகவேதான் தற்போது பாதிக்குப் பாதி பெண்களுக்கு அறுவை சிகிச்சை செய்யப்பட்டாலும், இறப்பு விகிதம் என்பது முழுவதுமாகக் குறைந்து விட்டது.

இயந்திரங்களுக்குக் கற்பிப்பது என்பது "விலங்குகள் எவ்வாறு கற்கின்றன" என்பதை அடிப்படையாக வைத்தே ஆராயப்பட்டது. இதில் பின்வரும் இரு பெரும் கோட்பாடுகள் முக்கியப் பங்கு வகிக்கின்றன.

<u>Bait's shyness:</u> தம்மை ஈர்க்குமாறு இருக்கும் உணவினைக் கண்டு அஞ்சுதல் என்பதே இதன் பொருள். அதாவது விஷம் தடவப்பட்ட ஆனால் பார்ப்பதற்குக் கவரக்கூடிய வகையில் இருக்கும் சுவையூட்டப்பட்ட உணவுகளைக் கண்டு எலிகள் அச்சம் கொள்ளும். எனவே அவற்றை முழுதாக உட்கொள்வதற்கு முன்னர், உணவின் ஒரு சிறு பகுதியை எடுத்து முதலில் சாப்பிடும். அதனால் தனக்கு யாதொரு பாதிப்பும் இல்லையெனில் உண்ணலாம் என்றும், பதிப்பு நேர்ந்தால் உண்ண வேண்டாம் என்றும் முடிவெடுக்கும். பின்னர் மீண்டும் அதே போன்ற ஒரு உணவினை மறுமுறை காணும்போது, தான் ஏற்கனவே நடத்திய சோதனை முடிவுகளின் அடிப்படையில் உண்ணலாமா வேண்டாமா என முடிவெடுக்கும். இதுவே இயந்திரவழிக் கற்றலிலும் நடைபெறுகிறது. மாபெரும் தரவுகளிலிருந்து ஒரு சிறுபகுதியை எடுத்து கணிணியானது முதலில் ஆராயும். இதுவே *sampling* எனப்படும். அச்சிறு பகுதி தரவுகள் *training data* என்று அழைக்கப்படும். அத்தரவுகளை உண்ணலாம் வேண்டாம் என்பது போன்று வகைப்படுத்துவதே *labeling* எனப்படும். இம்முடிவுகளை வைத்து வருகின்ற புதிய தரவுகளைக் கணிப்பது *Predicting the future data* எனப்படும். இதுபோன்று பல்வேறு பதங்கள் இயந்திரவழிக் கற்றலில் பயன்படுத்தப்படுகின்றன. ஆனால் இதுபோன்ற தீர்மானங்கள் சிலசமயம் தவறாக மாறிவிடவும் வாய்ப்புள்ளது. இதற்கு ஒரு சிறந்த உதாரணமாக பின்வரும் கோட்பாட்டைக் கூறலாம்.

<u>Pigeon's superstition:</u> புறாக்களின் தவறான கணிப்பு என்று நாம் இதைக் கூறலாம். ஒருமுறை *B.F.Skinner* எனும் மனோ தத்துவவியலாளர் புறாக்களை வைத்து ஆய்வு ஒன்றை நடத்தினார். அதில் பல புறாக்களை ஒன்றாக கூண்டிற்குள் வைத்து , அவைகளுக்கு குறிப்பிட்ட கால இடைவெளிகளில் உணவு சென்று சேருமாறு தானியங்கி ஒன்றை அமைத்தார். அதுவும் சரியாக செயல்பட்டு ஒவ்வொரு முறையும் உணவளித்து வந்தது. புறாக்கள் தனக்கு ஒவ்வொருமுறையும் உணவு எப்படி வருகிறது என்பதைக் கண்டுபிடிக்க அந்நேரத்தில் அது என்ன செய்துகொண்டிருந்தது என்பதை கவனிக்கத் தொடங்கியது. அதாவது ஒரு புறா தான் தலையசைக்கும்போதெல்லாம் உணவு வருவதால், தான் தலையசைப்பதால்தான் தனக்கு உணவு வருகிறதென்றும், மற்றொரு புறா அது குதித்துக்கொண்டிருக்கும்போதெல்லாம் உணவு வருவதால் அதனால்தான் உணவு வருகிறதென்றும் நினைத்துக் கொண்டது. ஆனால் இவையிரண்டும் தற்செயலாகத் தொடர்பான ஒன்றே! *(temporal correlation)* . உண்மையில் பார்த்தால் எந்தஒரு சம்மந்தமும் கிடையாது. ஆனால் புறாவோ இவ்விரண்டுக்கும் தவறான ஒரு தொடர்பினை உண்டாக்கி, அதனடிப்படையில் கணிப்பினை நிகழ்த்தி விடுகிறது. திடீரென தானியங்கி செயல்படும் நேரம் மாற்றப்பட்டு உணவு வரத் தொடங்கியது. ஆனால் இதையறியாப் புறாக்கள் தலையசைத்துப் பார்த்தும், குதித்துப் பார்த்தும் உணவு வராததால், அதன் எடை குறையத் தொடங்கியது. உணவு வருகின்ற நேரத்தை சரியாகக் கணிக்காததே இதற்குக் காரணம். இவ்வாறும் நமது இயந்திரங்களுக்கு நடந்துவிடக்கூடாது. தற்செயலாக நடைபெறும் தொடர்புடைய நிகழ்வுகளின் நிகழ்தகவு அதிகமாக இருந்தால் அதுவே நமது இயந்திரங்கள் நமக்கு அளிக்கும் கணிப்பாக இருந்துவிடக் கூடாது.

எலியினுடைய உதாரணத்தையே மீண்டும் பார்த்தால் அது சாப்பிட்டவுடன், ஒரு மின்சாரக் கம்பியில் அடிபட்டு பாதிப்படைகிறதெனில், இவ்விரண்டுக்கும் எந்த ஒரு தொடர்பும் இல்லையென்பது எலிக்குத் தெரியும். இதுபோன்று ஏலியிடம் காணப்படுகின்ற ஒரு அடிப்படையான அறிவையே நாம் கணினிக்குப் புகட்ட வேண்டும். இதுவே *Inductive bios* என்று அழைக்கப்படுகிறது. *Biosed* என்றால் பாரபட்சம் பார்ப்பது, ஒரு தலைப்பட்சமாக இருப்பது என்று பொருள். *Inductive bios* என்றால் இயந்திரத் தனமான முடிவுகளை அப்படியே ஏற்றுக்கொள்ளாமல் அறிவின் அடிப்படையில் பாரபட்சப் படுத்திப் பார்ப்பது என்று பொருள். இதுபோன்ற அறிவினை கணினிக்கு அளிப்பதற்கு அத்துறை சார்ந்த வல்லுநர்கள் தேவை. அவர்களே *domain expert* என்று அழைக்கப்படுகின்றனர்.

இயந்திர வழிக்கற்றலைப் பின்வரும் *4* விதங்களில் பிரிக்கலாம்.

1. Supervised vs Unsupervised Learning: ஒரு கணிப்பு நடைபெறுவதற்கு உள்ளீடு என்னவாக இருக்க வேண்டும்; வெளியீடு என்னவாக இருக்க வேண்டும்; இவ்விரண்டையும் எந்தெந்த விதிகளின்படி இணைக்க வேண்டும் போன்ற அனைத்தையும் சொல்லிக்கொடுத்துக் கண்காணிப்பது *supervised/structured learning* எனப்படும். உதாரணத்துக்கு நாம் கடைவீதிக்குச் சென்று வெண்டைக்காய் வாங்குவதற்கு முன், அதன் நிறத்தைப் பார்த்து அவ்வாறே அடிநுனியை லேசாகக் கிள்ளிப் பார்ப்போம். அது பசுமையாக, மிருதுவாக இருந்தால் வாங்கலாமென்றும், கடினமாக பழுப்பாக இருந்தால் வேண்டாமென்றும் முடிவு செய்வோம்.

- ஒரு வெண்டைக்காயை வாங்கலாமா வேண்டாமா என முடிவு செய்யும் காரணிகளான அதன் நிறம் தன்மை போன்றவை *domain set* எனப்படும். இவையே *X* எனும் உள்ளீட்டில் காணப்படும்.
- வாங்கலாம், வேண்டாம் எனும் மதிப்புகள் *labels* என்றழைக்கப்படும். இவையே *Y* எனும் வெளியீட்டில் அமையும்.
- ஒரு *mapping function* -ஆனது உள்ளீட்டில் உள்ள மதிப்புகளையும், வெளியீட்டில் உள்ள மதிப்புகளையும் தொடர்பு செய்யும். மென்மை -> வாங்கலாம், கடினம் -> வேண்டாம், பசுமை -> வாங்கலாம், பழுப்பு -> வேண்டாம். இதுவே *rules set* எனப்படும். *f: X -> Y*
- *Rules set* கற்பித்த விதிகளின் அடிப்படையில் கற்றுக்கொள்வது *learner* எனப்படும்.
- *Learner* கற்றுக்கொண்ட விஷயங்களின் அடிப்படையில் புதிதாக வருகின்ற உள்ளீடுகளுக்கு வெளியீடு என்னவாக இருக்கும் என முடிவு செய்வது *predictor* எனப்படும். அதாவது புதிதாக மற்றொரு வெண்டைக்காயைப் பார்க்கும்போது, அதனை மீண்டும் சோதனை செய்யத் தேவையில்லை. இந்த சோதனை முடிவுகளின் அடிப்படையிலேயே வாங்கலாம், வேண்டாம் என முடிவெடுக்கலாம்.

இதனை *classification* மற்றும் *regression* என்று 2 விதமாகப் பிரிக்கலாம். *Classification*-ல் மதிப்பு ஏதோ ஒரு வகையின் கீழ் அமையும். வெண்டைக்காய் உதாரணத்தில் வாங்கலாம் வேண்டாம் எனும் இரண்டு வகையின் கீழ் அமைவதை இதற்கு உதாரணமாகக் கொள்ளலாம். *Regression*-ன் மதிப்பு ஒரு உண்மையான முழு மதிப்பாக இருக்கும். வயிற்றிலுள்ள குழந்தையை *scan* செய்து ஆராய்ந்து அது பிறக்கும்போது எவ்வளவு எடை இருக்கும் என்பதை *kg*-ல் கணித்துக் கூறுவதை இதற்கு உதாரணமாகக் கொள்ளலாம்.

Unsupervised Learning - ல் வெறும் உள்ளீட்டுக்கான மதிப்புகள் மட்டுமே காணப்படும். வெளியீட்டுக்கான மதிப்பு என்னவாக இருக்குமென்றோ, அது எவ்விதிகளின்படி அமையுமென்றோ எந்தஒரு வரைமுறையும் கிடையாது. வெறும் உள்ளீட்டுக்கான மதிப்புகளை மட்டுமே ஆராய்ந்து, அதிலிருந்து ஒரு *pattern*-ஐக் கண்டுபிடித்து அதனை வெளியீட்டுக்கான கணிப்பாக நமக்கு வெளிப்படுத்தும். இதனை *clustering* மற்றும் *association* எனும் இரண்டு விதமாகப் பிரிக்கலாம். வாடிக்கையாளர்களின் விருப்பத்திற்கு ஏற்றார்போல் விற்பனையாகின்ற பொருட்களைக் கண்டுபிடித்து வகைப்படுத்துவதை *clustering*-க்கு உதாரணமாகக் கொள்ளலாம். இதில் விற்பனையாகின்ற தரவுகளை மட்டுமே உள்ளீடாக எடுத்துக்கொண்டு, அதன் போக்கிலேயே சென்று விற்பனையின் போக்கினைக் *(sales pattern)* கண்டுபிடிக்கும். அடுத்து இவ்வாறு கண்டுபிடிக்கப்பட்ட விவரங்களை எடுத்துக்கொண்டு, இதே மாதிரியான வேறு எந்தெந்த பொருட்களின் மீதெல்லாம் வாடிக்கையாளர்களுக்கு விருப்பம் தோன்றும் எனக் கணிப்பதை *association*-க்கு உதாரணமாகக் கொள்ளலாம். இதன் மூலம் ஒத்த விருப்பங்களின் கீழ் அமையும் பல்வேறு பொருட்களின் விற்பனையை நாம் அதிகரிக்கலாம். இதுவே *unsupervised/unstructured learning* எனப்படும்.

Structured மற்றும் *Unstructured* இவை இரண்டுக்கும் இடையில் அமைவது *semi-structured learning* எனப்படும். அதாவது ஒருசில தரவுகள் *label* செய்யப்பட்டும், மற்றவை *label* செய்யப்படாமலும் காணப்படும். உண்மையில் நிகழ்காலத்தில் வருகின்ற தரவுகள் நமக்கு இம்முறையில்தான் இருக்கும். பலகோடிக்கணக்கான தரவுகளை ஆராய்ந்து *label* செய்வது என்பது சத்தியமற்றது. அவ்வாறே அனைத்தையும் *label* செய்யாமல் விடுவதும் உதவாது. முக்கியமானவை *label* செய்யப்பட்டாக

வேண்டும். இதுபோன்று label செய்யப்பட்டும் செய்யப்படாமலும் இருக்கும் தரவுகளை நாம் மேற்கண்ட இரண்டு விதங்களிலும் ஆராயலாம். மனித/மிருக முகங்களின் கணிப்பு, குரல்களின் கணிப்பு போன்றவை இம்முறையில்தான் அமையும். Structured முறையில் label செய்யப்பட்டவற்றை மட்டும் training data-ஆகக் கொடுத்து, அதனடிப்படையில் மற்றவைகளைக் கணிக்கலாம். Unstructured முறைப்படி label செய்யப்பட்ட மற்றும் செய்யப்படாத அனைத்திலிருந்தும் ஒரு pattern-ஐக் கண்டுபிடித்து அதை வைத்தும் கணிக்கலாம்.

2. Passive vs Active Learning: *வருகின்ற தரவுகளை அப்படியே ஏற்றுக்கொண்டு, கொடுக்கப்பட்ட விதிகளின்படி ஆராய்ந்து கற்றுக்கொள்வது passive learning எனப்படும். ஒரு மின்னஞ்சல் spam-ஆ இல்லையா என சோதிப்பதை இதற்கு உதாரணமாகக் கொள்ளலாம். இதில் எவையெல்லாம் spam-ஐக் குறிக்கும் வார்த்தைகள் என்பது கணிணிக்குக் கற்பிக்கப்பட்டுவிடும். எனவே புதிதாக வருகின்ற ஒரு மின்னஞ்சல் இத்தகைய வார்த்தைகளில் ஏதேனும் ஒன்றைக் கொண்டிருந்தால் அதனை spam folder-க்கும், இல்லையெனில் inbox-க்கும் நகர்த்தும்.*

திடீரென spam-க்குரிய எந்த ஒரு வார்த்தையையும் கொண்டிராமல், ஆனால் சந்தேகத்திற்குரிய வழக்கத்திற்கு மாறாக ஒரு மின்னஞ்சல் வருகிறதெனில் (anamoly detection), தனது சந்தேகத்தைத் தீர்த்துக்கொள்ளும் பொருட்டு பல்வேறு கேள்விகளை எழுப்பி, அதற்கான விடைகளைப் பயனர்களிடமிருந்து பெற்றுக்கொண்டு அதனடிப்படையில் கற்றலைத் தொடங்குது active learning எனப்படும்.

3. Adversarial Teacher Method: *Spam filtering, malware detection, biometric recognition போன்றவற்றிலெல்லாம், ஆசிரியர் போன்று ஒருவர் செயல்பட்டு கொடுக்கப்பட்டுள்ள விதிகள் மீறப்படும்போதோ/சந்தேகிக்கப்படும்போதோ எது சரி எது தவறு என்பதை எடுத்துரைப்பார். வெறும் தரவுகளோடு சேர்த்து இம்முறையிலும் கணினிக்குக் கற்பிக்கப்படும்போது, காரண காரிய முறைப்படி பிரித்து கற்றுக்கொள்வதற்கான வாய்ப்பு அதற்குக் கிடைக்கிறது.*

4. Online vs Batch Learning: *நிமிடத்திற்கு நிமிடம் மாறுகின்ற தரவுகளைக் கண்காணித்து அதனடிப்படையில் கற்பது online learning எனவும், வரலாற்றுத் தரவுகளை எடுத்துக்கொண்டு அதனடிப்படையில் கற்பது batch learning எனவும் அழைக்கப்படும். Stock broker கணிக்கும் விதத்தை online-க்கு உதாரணமாகவும், மக்கள் தொகை கணிப்பு நடைபெறும் விதத்தை batch-க்கு உதாரணமாகவும் கொள்ளலாம். மக்கள் தொகை கணக்கெடுப்பில் 1970 – 80, 1981 – 90, 1991 – 2000, 2001 – 10 என்பது போன்று பல்வேறு பகுதிகளாகப் பிரிக்கப்பட்டு ஒவ்வொரு 10 வருடத்துக்கும் ஆய்வு நடைபெறுகிறது. இதனடிப்படையில் இனிவரும் வருடங்களுக்கான மக்கள் தொகை கணிப்பு நடைபெறும். இதுவே batch processing-க்கு சிறந்த உதாரணமாக அமையும்.*

இயந்திர வழிக்கற்றலில் உள்ள பல்வேறு கோட்பாடுகள் பற்றியும் அவற்றின்படி அமைந்த பல்வேறு வழிமுறைகள் (algorithmns) பற்றியும் இனிவரும் கட்டுரைகளில் காணலாம்.

Statistical Learning

புள்ளி விவரங்களைக் கொண்டு கற்பதே இயந்திர வழிக்கற்றலின் அடிப்படை. எந்த ஒரு கணிப்பும் தரவுகளாக அளிக்கப்படும் புள்ளி விவரங்களின் அடிப்படையிலேயே அமைகிறது. இத்தகைய புள்ளி விவரங்களைத் திறம்படக் கையாண்டு கணினிக்குக் கற்றுக் கொடுப்பது எப்படி என்று இப்பகுதியில் காணலாம். இதுவே Statistical learning model என்று அழைக்கப்படும்.

Domain set: *உள்ளீடாகத் தருகின்ற புள்ளி விவரங்களே இவ்வாறு அழைக்கப்படும். x={.....} என்பது domain set / instance space எனப்படும். இதிலுள்ள ஒவ்வொரு தனித்தனி விவரமும் domain points / instances எனும் பெயரில் அழைக்கப்படும்.*

$$X = x_1, x_2, \ldots\ldots, x_m$$

உதாரணத்துக்கு ஒரு *1000* பக்க நோட்டுப்புத்தகத்தின் விலையை எவ்வளவு வைக்கலாம் என ஒரு *algorithmn* மூலம் கணிப்பதற்கு, இதுவரை நாம் விலை நிர்ணயித்துள்ள நோட்டுப்புத்தகங்களின் பக்கங்கள் இதற்கு உள்ளீடாக அளிக்கப்படுகின்றன.

X = [10, 50, 150, ... 600, 800]

Label set: வெளியீடாக வர வேண்டிய விவரங்களை இது பெற்றிருக்கும். *Y = {........}* . உள்ளீடாக இருக்கின்ற தரவுகள் எந்தெந்த வகையின் கீழ் அமையும் எனும் மதிப்புகள் இதில் காணப்படும். இத்தகைய விவரங்களைத் தர உதவுபவர் *'domain expert'* என்று அழைக்கப்படுவார்.

$$Y = y_1, y_2, \ldots\ldots, y_m$$

உள்ளீட்டில் நாம் அளித்த நோட்டுப்புத்தகங்களின் விலைகள் இங்கு காணப்படும்.

Y = [50, 95, 250, ... 750, 999]

Mapping function: மேற்கூறிய இரண்டையும் வைத்துக்கொண்டு உள்ளீட்டுக்கும் வெளியீட்டுக்கும் இடையேயான தொடர்பை ஏற்படுத்தும் வேலையை *mapping function (f)* செய்கிறது. இதனை வைத்துத் தான் நமது *algorithm* நம்முடைய தரவுகளைப் பற்றிக் கற்றுக் கொள்கிறது.

$f : x \rightarrow y$

f : 10 பக்கங்கள் *-> 50* ரூபாய் *; 50* பக்கங்கள் *-> 95* ரூபாய் *; 150* பக்கங்கள் *-> 250* ரூபாய் *....*

Probability Distribution: நாம் கொடுக்கின்ற மாதிரித் தரவுகள் பரவலாக அமைய வேண்டும். வெறும் ஓரிரண்டு தரவுகளை மட்டும் கொடுத்துவிட்டு கணிப்புகள் நிகழ்த்தக் கூடாது. உதாரணத்துக்கு *10* பக்கம் அடுத்து *500* பக்கம் எனுமிரண்டு புத்தகத்தின் விலையை மட்டும் கொடுத்து விட்டு, திடீரென *1000* பக்க புத்தகத்தின் விலையை கணிக்கச் சொன்னால், அந்தக் கணிப்பு தவறாகத் தான் இருக்கும். இது ஓரளவுக்கு சரியாக இருக்க வேண்டுமானால், நாம் பயிற்சி அளிக்கப் பயன்படுத்தும் தரவுகளானது சீரான முறையில் பரவலாக அமைய வேண்டும். அதாவது *10, 50, 150 ....* என சீரான முறையில் பல்வேறு புத்தகங்களின் பக்கங்களும், அதற்கான விலைகளும் கொடுக்கப்பட வேண்டும். இதுவே *probability distribution* எனப்படும். இதனடிப்படையில் எடுக்கப்படும் முடிவே சரியானதாக இருக்கும்.

Sample data: நாம் தேர்ந்தெடுத்து அனுப்பும் மாதிரித் தரவுகளே *sample data* அல்லது *training data* எனப்படும். உதாரணத்துக்கு நம்மிடம் *500* புத்தகங்களின் பக்கங்களும், அதற்கான விலைகளும் இருக்கிறதெனில், அவை அனைத்தையும் கொடுத்துப் பழக்காமல், *0 - 50* பக்கங்கள் கொண்ட புத்தகத்திலிருந்து ஒரு புத்தகத்தின் விலை மற்றும் *50 - 100* பக்கங்கள் கொண்ட புத்தகத்திலிருந்து மற்றொரு புத்தகத்தின் விலை என்பது போன்று நாம் தேர்ந்தெடுத்து அனுப்பும் மாதிரித் தரவுகளே *Sample data* எனப்படும்.

Learner: நாம் தேர்ந்தெடுத்து அனுப்பியுள்ள மாதிரித் தரவுகளின் அடிப்படையில் புத்தகங்களின் விலையை நிர்ணயிப்பது பற்றிய அறிவை நமது *algorithm-* வளர்த்துக் கொள்கிறது. அவ்வாறு கற்றுக் கொண்ட *algorithm-*ஆனது *learner* என்று அழைக்கப்படுகிறது. *(A(S) = Algorithm of sample data)*

A (S)

$$S = X \times Y$$
$$S = (x_1, y_1), (x_2, y_2), \ldots\ldots, (x_m, y_m)$$

Predictor: *Learner* வளர்த்துக் கொண்ட அறிவின் மூலம், *label* செய்யப்படாத புதிய சில புள்ளி விவரங்கள் வரும்போது அவற்றையெல்லாம் எந்தெந்த *label-*ன் கீழ் அமைக்கலாம் என கணிப்பது *Predictor* எனப்படும். இதுவே *hypothesis / classifier* எனும் பல்வேறு பெயர்களில்

அழைக்கப்படுகின்றன. (h = hypothesis). அதாவது அதிகபட்சமாக 800 பக்கங்கள் கொண்ட புத்தகத்தின் விலை வரை மட்டும் தான் நமக்குத் தெரியுமெனில், அதற்கு மேல் பக்கங்கள் உயர உயர அதன் விலையை எவ்வளவு வைக்கலாம் என்பதை predictor கணித்துக் கூறும்.

$h : x \rightarrow y$

Validation data: ஒரு predictor-ன் கணிப்பு சரியாக உள்ளதா என சோதிப்பது observation எனவும், அதற்கு உதவும் தரவுகள் validation data என்றும் அழைக்கப்படும். ஒரு சிறந்த predictor-ஐத் தேர்வு செய்வதற்கு குறைந்தபட்சம் 30 observation-ஆவது தேவை. அதாவது நாம் 500 sample data-வைக் கையில் வைத்துள்ளோம் என்றால், அவை அனைத்தையும் கொடுத்து learner-ஐக் கற்பிக்காமல், வெறும் 300 தரவுகளை மட்டும் கொடுத்து கற்பிக்க வேண்டும். பின்னர் கற்றுக் கொடுத்த algorithm மூலம் மீதமுள்ள 200 தரவுகளுக்கான விலையை கணிக்கச் சொல்ல வேண்டும். இவ்வாறு ஒரு algorithm சரியாக கணிக்கிறதா இல்லையா என்பதை சோதிப்பதற்கு உதவும் அந்த 200 தரவுகளே validation data எனப்படும். பொதுவாக sample data-வின் 25% validation data-ஆக அமையும்.

Loss / Risk: கணிப்பு எப்போதும் 100% சரியாக அமையாது. அவ்வாறு அமைவது தவறும் கூட. predictor மூலம் எடுக்கப்படும் கணிப்பு உண்மையான மதிப்புடன் எந்த அளவு விகிதத்தில் வேறுபடுகிறது என்பதைக் கணக்கிட்டுக் கூறுவதே risk ஆகும். அதாவது நாம் validation data-ஐக் கொடுத்து சோதிக்கும் போது, நம் கையில் உள்ள உண்மையான மதிப்புக்கும், அதன் கணிப்புக்கும் உள்ள வேறுபாடே 'இழப்பு' எனப்படும். உதாரணத்துக்கு 850 பக்கங்கள் கொண்ட நோட்டுப்புத்தகத்தின் விலை 1200 ரூபாய் என நமக்கு ஏற்கனவே தெரிந்தாலும், ஒரு algorithm மூலம் கணிக்கப்படும் கணிப்பானது 1190 ரூபாய் அல்லது 1210 ரூபாய் என்றுதான் இருக்கும். ஏனெனில் இதுவரை அது கற்றுக்கொண்ட புத்தகங்களின் விலையை வைத்து தோராயமாக ஒரு விலையைக் கணிக்கும்போது, அது இவ்வாறுதான் இருக்கும். இதுவே சரியான முறையும் கூட. இந்த கடுகளவு வேறுபாடு இருந்தால்தான், நமது algorithm சரியாக வேலை செய்கிறது என்று அர்த்தம். விடையை மிகத் துல்லியமாகக் கொடுத்தால், அது தரவுகளைக் கற்றுக்கொண்டு கணிப்பு நிகழ்த்தாமல், மனப்பாடம் செய்து கொண்டு ஒப்பிக்கிறது என்றே அர்த்தம் (இதனைப் பற்றி over-fitting-ல் காணலாம்). இதுபோன்ற Risk-ஐ அளவிடுவதில் 2 படிகள் உள்ளன. முதலில் true risk அடுத்து empirical risk.

True Risk : 1200 ரூபாய் மதிப்பு கொண்ட நோட்டுப்புத்தகத்தின் விலையானது 1190 ரூபாய் என கணிக்கப்படும்போது, இடையிலுள்ள 10 ரூபாய் என்பதுதான் உண்மையான risk. இதனை generalization error என்றும் கூறுவர். பொதுப்படையான ஒன்றை உருவாக்குவதால் ஏற்படுகின்ற பிழை எனப் பொருள். ஆனால் இது போன்ற பிழைகளை ஒவ்வொரு தரவிற்க்கும் தனித்தனியாகக் கணக்கிட்டுக் கூறுவதற்கு பதிலாக அவைகளின் சராசரியான empirical risk என்ற ஒன்று கண்டறியப்படுகிறது. R(h) என்பது Risk of hypothesis ஆகும். அதாவது கணிப்பின் மூலம் எடுக்கப்பட்ட வெளியீடும் h(x), mapping மூலம் அமைய வேண்டிய உண்மையான வெளியீடும் f(x) சமமாக இல்லாத பட்சத்தில், அது ஒரு Risk என்பதையே கீழ்க்கண்ட சூத்திரம் கூறுகிறது.

$$\begin{aligned} R(h) &= L(h(x_i), f(x_i)) \\ &= h(x_i) \neq f(x_i) \end{aligned}$$

Empirical risk: சோதனைக்காக நாம் 200 புத்தகங்களை அளிப்பதாக வைத்துக் கொண்டால், அவை ஒவ்வொன்றின் உண்மையான விலைக்கும் - கணிக்கப்படுகின்ற விலைக்குமான வேறுபாட்டைக் கண்டறிந்து, அவற்றை சராசரி எடுப்பதன் மூலம் அனைத்துத் தரவுகளுக்குமான தோராயமான risk ஒன்றை அமைக்கலாம். இதுவே empirical risk ஆகும். இதன் மூலம் கணிக்கப்படுகின்ற புத்தகங்களின் விலையானது தோராயமாக இவ்வளவு ரூபாய் வேறுபாட்டில் அமைந்திருக்கும் என நம்மால் வரையறுத்துக் கூற முடியும்.

$$R_{emp}(h) = \frac{1}{m}\sum_{1}^{m} h(x_i) \neq f(x_i)$$

Empirical Risk Minimization : நமது சோதனைக்காக பல்வேறு *algorithms* கொண்டு உருவாக்கப்பட்ட பல்வேறு *predictor*-களில் சிறந்ததைக் கண்டறிய *validation data*-வைக் கொடுத்து ஒவ்வொரு கணிப்பானும் ஆராயப்படுகிறது. பல்வேறு *observations* மூலம் ஒவ்வொன்றிலும் உள்ள இழப்பின் சராசரியானது கண்டுபிடிக்கப்படுகிறது. இத்தகைய சராசரி இழப்பின் மதிப்புகளில் எதனுடைய மதிப்பு மிகக் குறைவாக உள்ளதோ அதனைக் கண்டுபிடிப்பதே *Empirical Risk Minimization* எனப்படும். இதற்கான சூத்திரம் பின்வருமாறு. இதில் *arg min* என்பது *argument of minimum* எனப் பொருள்படும். இவ்வாறு கண்டறியப்பட்ட மதிப்பினை, ஒவ்வொரு *algorithm*-ம் தனக்கென்றே கொண்டுள்ள ஒருசில *parameters* கொண்டு அளவிடுகிறது. அதன் பின்னரே இந்தக் கணிப்பினை எவ்வளவு தூரம் நம்பலாம், இதனால் கணிக்கப்படும் மதிப்பு எந்த அளவுக்குத் துல்லியமாக இருக்கும் என்பது போன்ற விஷயங்களெல்லாம் கணக்கிட்டுக் கூறுகிறது. இதைப் பற்றி *PAC Model*-ல் காணலாம்.

$$ERM = arg \min_{h \in H} R_{emp}(h)$$

Probably Approximately Correct (PAC Method)

ஒரு கணிப்பான் மூலம் நிகழ்த்தப்படும் கணிப்பு எவ்வளவு தூரம் சரியானதாக இருக்கும், அதனை எவ்வளவு தூரம் நம்பலாம் என்பது போன்ற விஷயங்கள் எல்லாம் இந்த முறையில் கணக்கிடப்படுகிறது. முதலில் ஒரு கணிப்பானின் கணிப்பு *probably approximately correct* -ஆக அமைவதற்கு அவற்றில் என்னென்ன பண்புகளெல்லாம் இருக்க வேண்டும் என்பதை ஒருசில வரையறைகள் கொண்டு சோதிக்கிறது. அதாவது *over-fitting* இல்லாமல் இருக்கிறதா, *inductive bias* பெற்று விளங்குகிறதா,*i.i.d* முறையில் பயிற்சித் தரவுகள் அளிக்கப்பட்டுள்ளதா,அதன் *sample complexity* எவ்வளவு இருந்தால், கணிப்பு ஓரளவுக்கு சரியாக அமையும் என்பது போன்ற நோக்கில் எல்லாம் ஆராயப்படுகிறது. பின்னர் *accuracy* மற்றும் *confidence parameters* மூலம் நமது கணிப்பு எவ்வளவு தூரம் துல்லியமானது என்பதைக் கணக்கிடுகிறது. இம்முறையில் *realizability assumption* எனும் அனுமானம் காணப்படும். ஆனால் இது நாம் அடுத்து காணப்போகும் *Agnostic PAC Model*-ல் நீங்கிவிடும். இங்கு குறிப்பிட்டுள்ள ஒவ்வொன்றின் விளக்கத்தையும் கீழே காணலாம்.

Overfitting: ஒருசில மாதிரித் தரவுகளைக் கொடுத்து *learner*-ஐப் பழக்காமல், ஒட்டுமொத்தமாக அனைத்துத் தரவுகளையும் கொடுத்துப் பழக்கினால் *overfitting* என்ற அபாயம் ஏற்பட வாய்ப்பு உள்ளது. இவ்வாறு அளவுக்கு அதிகமாகத் தரவுகளைப் பெற்றுக்கொள்ளும் *learner*-ஆனது கற்றுக்கொள்ள முயற்சி செய்யாமல், சுலபமாக மனப்பாடம் செய்துவிடுகிறது. சோதனையின்போதும், நாம் எதிர்பார்க்கின்ற மதிப்பினைத் துல்லியமாக அளிக்கின்றது. இதில் உள்ள *risk*-ன் மதிப்பு எப்போதும் குறைவே. அதனாலேயே இதை ஒரு சரியான கணிப்பாக எடுத்துக் கொள்ள முடியாது. ஏனெனில் பயிற்சியின் போது அளிக்கப்படாத புதிய தரவுகளுக்கு இதனால் முறையாக கணிப்பினை நிகழ்த்த முடியாது. ஆகவே இந்த *Overfitting*-ஐ இல்லாமல் செய்வதற்காக உள்ளதே *inductive bias* ஆகும்.

Inductive bias: *hypothesis class* என்பது மாதிரித் தரவுகளில் உள்ள ஒவ்வொன்றையும் எந்தெந்த *label*-வுடன் முறைப்படுத்திக் கற்க வேண்டும் என்ற தொடர்பினை விளக்குகிறது. இதுவே *Inductive bias* ஆகும். *biased* என்றால் ஒன்றினைச் சார்ந்திருப்பது என்று பொருள். இம்முறையில் *learner*-ஆனது,

hypothesis class-ல் கூறப்படுகின்ற தொடர்புகளின் அடிப்படையில், தரவுகளைப் பற்றிய அறிவை வளர்த்துக்கொள்கிறது. அவ்வாறு பெற்றுக்கொண்ட அறிவினடிப்படையில் கணிப்பினை நிகழ்த்துவதே inductive bias என்றழைக்கப்படுகிறது. இதுவே சரியான முறையும் கூட!

Hypothesis Class: *ஒரு learner-ஐ inductive bias-ஆக இருக்குமாறு அமைக்க உதவுவது hypothesis class ஆகும். இதனை finite & infinite என்று இரண்டு வகையாகப் பிரிக்கலாம். Hypothesis என்பதை தமிழில் கருதுகோள் எனச் சொல்லலாம். என்னென்ன கணிப்புகளின் கீழ் உள்ளீடுகள் இருக்கும் எனும் வரையறையைக் கொடுத்து, அதன்கீழ் கணிக்கச் சொல்லுவது finite hypothesis class. உதாரணத்துக்கு youtube-ல் login செய்யும் ஒருவர் காலையில் பக்திப் பாடலும், மலையில் இளையராஜா பாடலும் தொடர்ச்சியாக கேட்டுக் கொண்டிருக்கிறார் எனில், அவருக்கான hypothesis class பக்திப் பாடல் மற்றும் இளையராஜா பாடல் எனும் இரண்டு வகையின் கீழ் அமையும். இதனை finite hypothesis class-க்கு உதாரணமாகச் சொல்லலாம். ஆனால் மற்றொருவரோ எந்த வகையின் கீழ் அவருடைய ரசனை இருக்கும் என வரையறுக்கவே முடியாத அளவுக்கு, காதல், பக்தி, நகைச்சுவை, சண்டை, நடனம், குழந்தைப் பாடல்கள் என பல்வேறு வகையிலிருந்து மாற்றி மாற்றிப் பார்க்கிறார். எனவே, அவருக்கான hypothesis class-ல் இவ்வளவு வகைகள் தான் இருக்கும் என வரையறுக்கவே முடியாத படி நீண்டு கொண்டே செல்லும். இதையே infinite hypothesis class-க்கு உதாரணமாகச் சொல்லலாம்.*

Sample complexity : *மாதிரித் தரவுகளின் எண்ணிக்கை மிகவும் குறைந்து இருந்தாலோ அல்லது அளவுக்கு அதிகமாக இருந்தாலோ கணிப்பு சரியாக நடைபெறாது. எனவே ஒரு கணிப்பானின் வெற்றியானது அதற்கு மாதிரியாக கொடுக்கப்படுகின்ற தரவுகளின் எண்ணிக்கையைப் பொறுத்தே அமைகிறது. தோராயமாக எவ்வளவு மாதிரித் தரவுகள் கொடுத்தால், அதனுடைய கணிப்பு ஓரளவுக்கு சரியாக இருக்கும் எனக் கூறுவதே sample complexity ஆகும்.*

$$S \sim D^m$$

S similarity D to the power of m என்பதில் m -ஆனது மாதிரியாக எடுக்கப்படும் தரவுகளின் எண்ணிக்கை ஆகும். அந்த எண்ணிக்கையைக் கணக்கிட உதவும் கிளைத் தேற்றம் பின்வருமாறு.

$$m \leq \left[\frac{\log |H| / \delta}{\epsilon} \right]$$

இதற்கு அளிக்கப்படும் மாதிரித் தரவுகளானது i.i.d எனும் அனுமானத்தின் வழியே நடக்கிறது. i.i.d என்றால் independently identically distributed என்று பொருள். ஒன்றோடொன்று சார்பற்ற தனித்தனியான தரவு மாதிரிகளை எடுத்தனுப்பி learner-க்குக் கற்பிப்பதையே இது வலியுறுத்துகிறது.

Realilzability assumption: *நாம் ஏற்கனவே கண்ட உதாரணத்தில், புத்தகங்களின் பக்கங்கள் அதிகரித்தால் அதனுடைய விலையும் அதிகரிக்கும் எனும் அனுமானத்தினை நமது algorithm வளர்த்துக் கொள்கிறது. இதுவே realizability assumption எனப்படும். ஆனால் இந்த அனுமானம் எல்லா வகையான கணிப்புக்கும் பொருந்தாது. உதாரணத்துக்கு ஒரு நாணயத்தை சுண்டி விட்டால், தலை விழுமா பூ விழுமா என்பதற்கு எந்த ஒரு அனுமானமும் செய்ய முடியாது. இதுபோன்ற நிலையற்ற தன்மையைக் குறிக்கும் கணிப்புகளைப் பற்றி Agnostic PAC model-ல் காணலாம்.*

Accuracy parameter: *ஒரு predictor/classifier-ன் மதிப்பு எவ்வளவு தூரம் துல்லியமாக இருக்கும் என்பதைக் குறிக்க ∈ எனும் குறியீடு பயன்படுகிறது. எனவே$R(h) > \in$ என்பது ஒரு கணிப்பானின் தோல்வியாகவும், $R(h) <= \in$ என்பது தோராயமாக ஒரு நல்ல கணிப்பானாகவும் எடுத்துக் கொள்ளப்படுகிறது*

Confidence parameter: *இது delta மதிப்பின் அடிப்படையில் குறிக்கப்படுகிறது.*

$$\delta = \begin{cases} 1, & h(x_i) = f(x_i) \\ 0, & h(x_i) \neq f(x_i) \end{cases}$$

இதில் நாம் எதிர்பார்ப்பதும், கணிப்பான் எடுத்துக்கூறுவதும் சரியாக இருப்பதற்கான நிகழ்தகவு 1 எனவும், தவறாக அமைவதற்கான நிகழ்தகவு 0 எனவும் கொள்ளப்படுகிறது. இதன் அடிப்படையில் பார்த்தால் 1 என்பது இரண்டும் சமமாக அமைவதற்கான நிகழ்தகவு எனக் கொண்டால், 1-δ என்பது உண்மையான கணிப்பினை எடுத்துக்கூறப் போதுமானதாக இல்லாமல் அமைவதற்கான நிகழ்தகவு ஆகும். இதுவே இத்தகைய மாதிரிகளை எவ்வளவு தூரம் நம்பலாம் என்பதைக் குறிக்கும் confidence parameter (1-δ) ஆகும்.

அடுத்ததாக இதுவரை நாம் கற்றுக்கொண்ட விஷயங்களை வைத்து simple linear regression-ஐ உருவாக்குவது எப்படி என்று பார்க்கலாம்.

Linear Regression

Simple Linear Regression

Simple Linear என்பது இயந்திர வழிக் கற்றலில் உள்ள ஒரு அடிப்படையான algorithm ஆகும். இதில் இரண்டு விவரங்கள் எவ்வாறு தொடர்பு படுத்தப்படுகின்றன, algorithm எவ்வாறு தனது புரிதலை மேற்கொள்கிறது, அந்தப் புரிதல் எந்த அளவுக்கு சரியாக உள்ளது என்பது போன்ற விஷயங்களையெல்லாம் ஒருசில தரவுகளை வைத்து செயல்முறையில் செய்து பார்க்கப் போகிறோம். உதாரணத்துக்கு ஒரு பிட்சாவின் அளவினைக் கொண்டு அதன் விலையை எவ்வாறு நிர்ணயிப்பது என இப்பகுதியில் காணலாம். இதுவரை நம்மிடமுள்ள அனைத்து பிட்சாவின் அளவும், அதற்கான விலைகளும் X மற்றும் Y variable-ல் எடுத்துக்கொள்ள வேண்டும். இதுவே label set மற்றும் domain set ஆகும்.

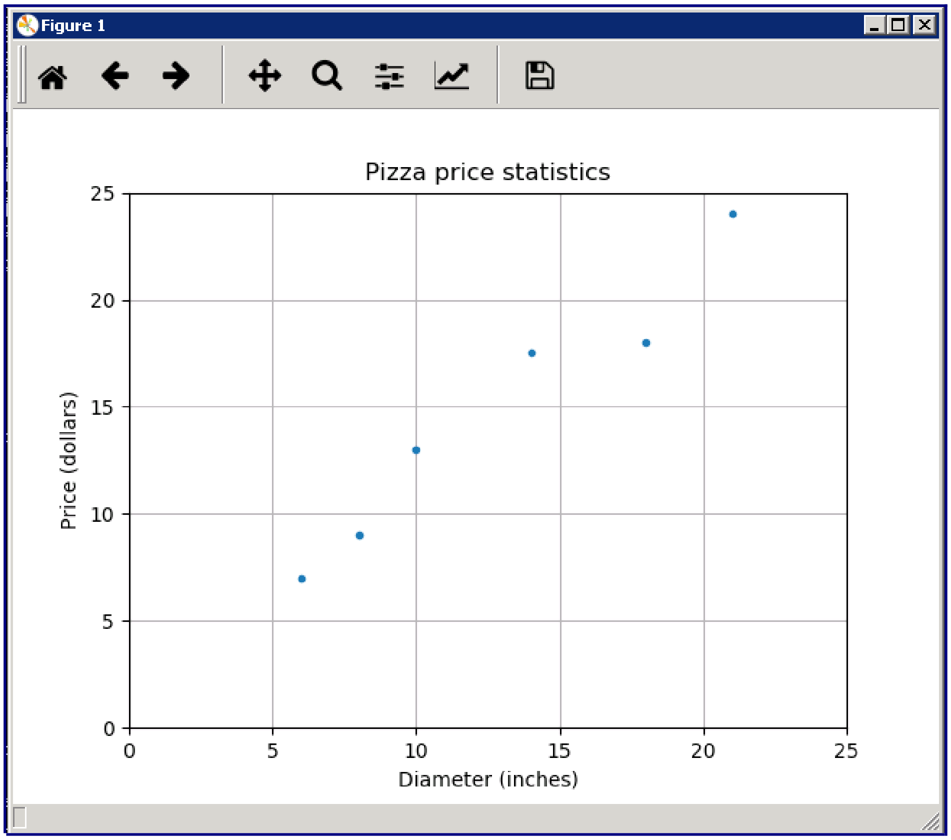
x=[6,8,10,14,18,21]
y=[7,9,13,17.5,18,24]

பல்வேறு பிட்சாவின் விட்டத்தைப் (in inch) பெற்றிருக்கும் X என்பது explanatory variable எனவும், அவற்றினுடைய விலைகளைக்(in dollar) கொண்டிருக்கும் Y என்பது response variable எனவும் அழைக்கப்படும். புள்ளி விவரங்களாக இருக்கும் இவற்றை ஒரு வரைபடமாக வரைந்து பார்ப்போம். அப்போதுதான் அவை செல்லும்போக்கு நமக்குத் தெரியும். matplotlib என்பது வரைபடங்களை வரைந்து காட்ட உதவும் ஒரு library ஆகும். இதிலுள்ள pyplot மூலம் நமது புள்ளி விவரங்களுக்கான வரைபடம் வரையப்பட்டுள்ளது.இதற்கான நிரல் பின்வருமாறு.

https://gist.github.com/nithyadurai87/cb77831526033da63be0790f917efe63

```
import matplotlib.pyplot as plt

x=[[6],[8],[10],[14],[18],[21]]
y=[[7],[9],[13],[17.5],[18],[24]]

plt.figure()
plt.title('Pizza price statistics')
plt.xlabel('Diameter (inches)')
plt.ylabel('Price (dollars)')
plt.plot(x,y,'.')
plt.axis([0,25,0,25])
plt.grid(True)
plt.show()
```

இது வெளிப்படுத்துகின்ற வரைபடம் பின்வருமாறு இருக்கும்.



இந்த வரைபடத்தில் பிட்சாவின் விட்டத்துக்கும், அதன் விலைக்குமிடையே நேர்மாறல் தொடர்பு இருப்பதைக் காணலாம். அதாவது ஒன்றின் மதிப்பு அதிகரிக்க அதிகரிக்க மற்றொன்றும் அதிகரிக்கும் என்பதே நேர்மாறல். இங்கும் அப்படித்தான் உள்ளது. அடுத்து இதை வைத்து ஒரு *algorithm*-க்குக் கற்றுக் கொடுப்பதற்கான நிரல் பின்வருமாறு.

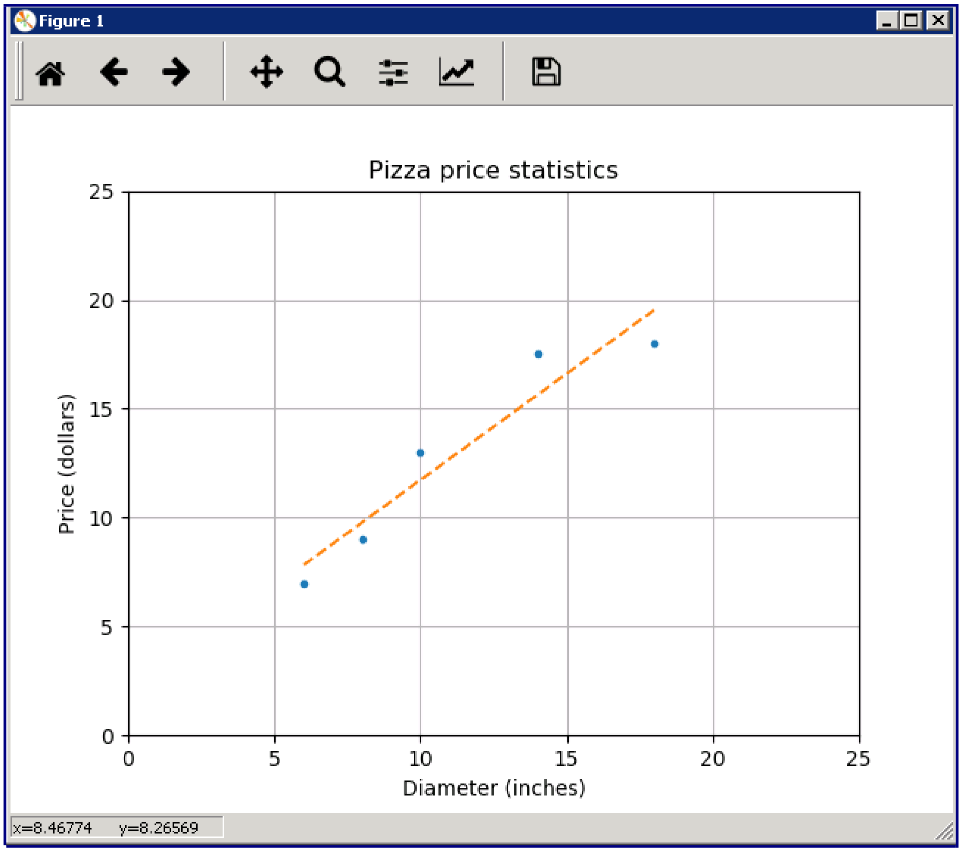
https://gist.github.com/nithyadurai87/d94507f9052a6120dce5f20e31806cea

```
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

x = [[6], [8], [10], [14], [18]]
y = [[7], [9], [13], [17.5], [18]]
model = LinearRegression()
model.fit(x,y)

plt.figure()
plt.title('Pizza price statistics')
plt.xlabel('Diameter (inches)')
plt.ylabel('Price (dollars)')
plt.plot(x,y,'.')
```

```
plt.plot(x,model.predict(x),'--')
plt.axis([0,25,0,25])
plt.grid(True)
plt.show()

print ("Predicted price = ",model.predict([[21]]))
```

__நிரலுக்கான விளக்கம்:__ *sklearn என்பது பல்வேறு வகையான algorithms-ஐக் கொண்ட ஒரு package ஆகும். இதிலிருந்து linear_model library-ல் உள்ள LinearRegression() class-ஆனது import செய்யப்படுகிறது. இதுவே கணிப்பான்/predictor ஆகும். இதற்கு நமது தரவுகளைப் பற்றிக் கற்றுக் கொடுப்பதற்காக fit() எனும் method பயன்படுத்தப்பட்டுள்ளது. பிறகு நமது Model எவ்வாறு கற்றுக் கொண்டுள்ளது என்பதை அறிய pyplot மூலம் வரைபடம் வரைந்து காட்டப்பட்டுள்ளது. கடைசியாக predict() எனும் function, நமது model-ன் மீது செயல்பட்டு 21 inch அளவு கொண்ட பிட்சாவின் விலை எவ்வளவு இருக்கும் என கணிக்கிறது.*

__நிரலுக்கான வெளியீடு:__ *மேற்கண்ட பைத்தான் நிரலை இயக்கும்போது, பின்வருமாறு ஒரு வரைபடம் வெளிப்படுகிறது. பின்னர் 21 inch அளவு கொண்ட பிட்சாவின் விலையாக [[22.46767241]] எனும் மதிப்பினை வெளிப்படுத்துகிறது.*



இந்த வரைபடத்தில் அனைத்துப் புள்ளி விவரங்களுக்கும் மத்தியில் ஒரு நேர்கோடு உள்ளத்தைக் காணலாம். இதுவே hyperplane என்று அழைக்கப்படும்.. அதாவது இந்தக் கோடுதான் algorithm-ன் புரிதல். இனிவரும் பிட்சாவின் விட்டத்துக்கு இந்தக் கோட்டினை வைத்துத்தான் விலையைக் கணிக்கும். இந்தப் புரிதலுக்கான கோட்டிற்கும் உண்மையான புள்ளி விவரங்கள் அமைந்துள்ள இடத்திற்குமிடையே ஒரு சிறு இடைவெளி இருப்பதைக் காணலாம். இந்த இடைவெளியே residuals அல்லது training error என்று அழைக்கப்படும். இங்கு நாம் கண்ட உதாரணத்தில், 21 inch விட்டம் கொண்ட பிட்சாவின் விலை 24 டாலர் என நமக்கு ஏற்கனவே தெரியும். ஆனால் இதையே நமது Model

கொண்டு கணிக்கும்போது அதன் விலை 22 டாலர் எனக் காட்டுவதைக் காணலாம். இதையே *generalization error / risk* என்றும் கூறுவர். அதாவது பொதுப்படையாக ஒரு புரிதலை அமைத்துக்கொண்டு, அதை வைத்து கணிப்பதால் ஏற்படும் *error* என்று பொருள். *Residual sum of squares* என்பது இந்த *risk*-ஐக் கணக்கிட உதவும் ஒரு *function* ஆகும். இதையே *loss function / cost function* என்று கூறுவர். *Residuals sum of squares* என்பது இத்தகைய இழப்பின் சராசரியை கண்டறிந்து கூறும். இந்த *risk*-க்கு என்ன காரணம், இதை எப்படிக் கணக்கிடுவது, கண்டுபிடிப்பது என்பது பற்றிப் பின்வருமாறு காணலாம்.

<u>Algorithm - Simple linear:</u>

நமது கணிப்பான் *fit()* மூலம் கற்றுக் கொள்ளும் சமன்பாடு பின்வருமாறு இருக்கும். இதுவே *Simple linear regression*-க்கான *algorithm* ஆகும்.

$y = \alpha + \beta x$

இதில் நமது *explanatory, response variables* தவிர்த்து, α *(intercept term)*, β *(coefficient)* எனும் இரண்டு *parameters* காணப்படுகின்றன. அதாவது இவற்றையும் சேர்த்தே நமது *algorithm* கற்றுக் கொள்கிறது. இதுவே நமது *model*-ன் *risk*-க்குக் காரணம். இதைக் கண்டுபிடித்து விட்டால் *risk*-ஐ எப்படிக் குறைப்பது என்பது தெரிந்துவிடும். முதலில் β-ன் மதிப்பினைக் கண்டுபிடிக்க வேண்டும். பின்னர் இதை வைத்து α-ன் மதிப்பினைக் கண்டுபிடித்து விடலாம். *Variance* என்பது நம்முடைய *explanatory variable*-ல் உள்ள தரவுகளானது எவ்வளவு இடைவெளி வித்தியாசத்தில் அமைந்துள்ளது என்பதைக் குறிக்கும்.. *[1,3,5,7,9,11......]* என்று இருக்கும் பட்சத்தில் அதன் *variance 0* ஆகும். ஏனெனில் இவை சீரான இடைவெளியுடன் அமைந்துள்ளது.. அதுவே *[1,5,7,10,11.....]* என்று அடுத்தடுத்த எண்களுக்கான இடைவெளி சீரற்று இருக்கும் பட்சத்தில், அந்த சீரற்ற தன்மை எவ்வளவு இருக்கிறது எனக் கணக்கிடுவதே *variance* ஆகும். *Co-variance* என்பது நமது *explanatory & response variables* இரண்டும் சேர்ந்து எவ்வளவு இடைவெளி வித்தியாசத்தில் அமைந்துள்ளது என்பதைக் குறிக்கும்.. இவ்விரண்டுக்கும் இடையே *linear* தொடர்பு இல்லையென்றால், இதன் மதிப்பு *0* ஆகும். இவைகளுக்கான சூத்திரங்கள் பின்வருமாறு.

$$y = \alpha + \beta x$$

$$\alpha = \bar{y} - \beta\bar{x}$$

$$\beta = \frac{\operatorname{cov}(x, y)}{\operatorname{var}(x)}$$

$$\operatorname{var}(x) = \frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1}$$

$$\operatorname{cov}(x, y) = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{n-1}$$

Numpy library-ல் உள்ள ஒருசில கணித *functions* மேற்கண்ட சூத்திரங்களின் படி நமது தரவுகளைப் பொருத்தி விடையை அளிக்கும் வேலையைச் செய்கின்றன.

https://gist.github.com/nithyadurai87/406747e718d04a4bc339f740b5f9de62

```
from sklearn.linear_model import LinearRegression
import numpy as np
```

```
x = [[6], [8], [10], [14], [18]]
y = [[7], [9], [13], [17.5], [18]]
model = LinearRegression()
model.fit(x,y)

print ("Residual sum of squares = ",np.mean((model.predict(x)- y) ** 2))
print ("Variance = ",np.var([6, 8, 10, 14, 18], ddof=1))
print ("Co-variance = ",np.cov([6, 8, 10, 14, 18], [7, 9, 13, 17.5, 18])[0][1])
print ("X_Mean = ",np.mean(x))
print ("Y_Mean = ",np.mean(y))
```

இதன் வெளியீடாக பின்வரும் மதிப்புகள் வெளிப்படும்.

Residual sum of squares = 1.7495689655172406
Variance = 23.2
Co-variance = 22.650000000000002
X_Mean = 11.2
Y_Mean = 12.9

இவ்வாறு நாம் கண்டுபிடித்த மதிப்புகளை சமன்பாட்டில் பொருத்தினால், *21 inch* விட்டம் கொண்ட பிட்சாவின் விலை எவ்வாறு *22.46* டாலர் எனக் காட்டுகிறது என்பதை அறியலாம்.

$y = \alpha + \beta x$

$= \alpha + \beta\ (21)$

$= 1.92 + (0.98*21) = 1.92 + 20.58 = 22.5$

where as,

$\beta = 22.65 / 23.2 = 0.98$

$\alpha = 12.9 - (0.98*11.2) = 12.9 - 10.976 = 1.92$

R-squared Score: கடைசியாக நாம் உருவாக்கியுள்ள *model* எவ்வளவு தூரம் உண்மையான மதிப்பினை அளிக்கும் அளவுக்குப் பொருந்தியுள்ளது என்பதைக் கணக்கிடுவதே *R-Squared score* ஆகும்.

https://gist.github.com/nithyadurai87/a39ecee72dc4a266933621c298e80df9

```
from sklearn.linear_model import LinearRegression
import numpy as np
from numpy.linalg import inv,lstsq
from numpy import dot, transpose

x = [[6], [8], [10], [14], [18]]
y = [[7], [9], [13], [17.5], [18]]
model = LinearRegression()
model.fit(x,y)

x_test = [[8], [9], [11], [16], [12]]
y_test = [[11], [8.5], [15], [18], [11]]
print ("Score = ",model.score(x_test, y_test))
```

R-squared score = 0.6620052929422553

score() எனும் *function,* அதற்கான சூத்திரத்தில் , நமது *validation data-*வைப் பொருத்தி விடையினை

நமக்கு அளிக்கிறது.. பொதுவாக *score* வெளிப்படுத்தும் மதிப்பு *0*-லிருந்து *1*-வரை அமையும். *1* என்பது *overfit*-ஆதலால், சற்று *1*-க்கு நெருங்கிய மதிப்பாக இருந்தால், இதனை நாம் ஏற்றுக்கொள்ளலாம். இங்கு நம்முடைய *model*-ன் மதிப்பு *0.66* என வெளிப்பட்டுள்ளது. *Simple linear* -ஐ விட *multiple linear*-ல் *accuracy* இன்னும் அதிகமாக இருக்கும். இதைப்பற்றி அடுத்து காண்போம்.

Multiple Linear Regression

Simple linear-ல் ஒரு பிட்சாவின் விலையானது அதன் விட்டதைப் பொறுத்து அதிகரிப்பதைக் கண்டோம். ஆனால் உண்மையில் விலை அதிகரிப்புக்கு அதன் மீது தூவப்படும் *toppings*-ம் ஒரு காரணியாக இருக்கும். எனவே ஒரு பிட்சாவின் விலை அதன் விட்டம் மற்றும் அதிலுள்ள *toppings*-ன் எண்ணிக்கை ஆகிய இரண்டையும் பொறுத்து அமைகிறது. இதுபோன்று ஒன்றுக்கும் மேற்பட்ட *explanatory variables*-ஐப் பொறுத்து, அதன் *response variable* அமைந்தால், அதுவே *multiple linear regression* எனப்படும். இதற்கான சமன்பாடு பின்வருமாறு இருக்கும்.

$$y = \alpha + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n$$

$$\begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_n \end{bmatrix} = \begin{bmatrix} \alpha & + & \beta X_1 \\ \alpha & + & \beta X_2 \\ & \vdots & \\ \alpha & + & \beta X_n \end{bmatrix}$$

$$\beta = \left(X^T X\right)^{-1} X^T Y$$

மேற்கண்ட அதே உதாரணத்தில் *explanatory variable*-வுடன் *toppings*-ன் எண்ணிக்கையும் சேர்த்து *multiple linear*-ஐ உருவாக்கியுள்ளோம். இது பின்வருமாறு.

https://gist.github.com/nithyadurai87/7068c32bd4d7fccb67ccca39623f68bc

```
from sklearn.linear_model import LinearRegression
from numpy.linalg import lstsq
import numpy as np

x = [[6, 2], [8, 1], [10, 0], [14, 2], [18, 0]]
y = [[7], [9], [13], [17.5], [18]]
model = LinearRegression()
model.fit(x,y)

x1 = [[8, 2], [9, 0], [11, 2], [16, 2], [12, 0]]
y1 = [[11], [8.5], [15], [18], [11]]

predictions = model.predict([[8, 2], [9, 0], [12, 0]])
print ("values of Predictions: ",predictions)
print ("values of β1, β2: ",lstsq(x, y, rcond=None)[0])
print ("Score = ",model.score(x1, y1))
```

<u>நிரலுக்கான வெளியீடு:</u>

மேற்கண்ட நிரலுக்கான வெளியீடு பின்வருமாறு அமையும். இதில் *accuracy* அதிகரித்திருப்பதைக் காணலாம். *simple linear*-ல் *0.66* என்றால் *multiple linear*-ல் *0.77* என இருப்பதை

கவனிக்கவும். எப்போதும் *simple linear*-ஐ விட *multiple linear*-ஐப் பயன்படுத்தும்போது *accuracy* இன்னும் அதிகமாக இருக்கும்

values of Predictions: [[10.0625]
[10.28125]
[13.3125]]
values of ß1, ß2: [[1.08548851]
[0.65517241]]
Score = 0.7701677731318468

நாம் கண்டுபிடித்த இந்த மதிப்புகள் மேற்கண்ட சமன்பாடில் பின்வருமாறு பொருந்துகின்றன. இதில் *intercept term* -ஆன α-ன் மதிப்பு *x1* மற்றும் *x2* எனும் இரண்டு *variables*-ஐயும் பொறுத்து அமைவதால், இது பொதுவாக ஒரு *constant*-ஆக இருக்கும்.

$10.06 = \alpha + (1.09 * 8) + (0.66 * 2)$
$= \alpha + 8.72 + 1.32$
$= \alpha + 10.04$

$10.28 = \alpha + (1.09 * 9) + (0.66 * 0)$
$= \alpha + 9.81 + 0$
$= \alpha + 9.81$

$13.31 = \alpha + (1.09 * 12) + (0.66 * 0)$
$= \alpha + 13.08 + 0$
$= \alpha + 13.08$

Simple Linear Algorithm

Simple linear regression -க்கான சமன்பாடு பின்வருமாறு அமையும். இதை வைத்து *(1,1) , (2,2) , (3,3)* எனும் புள்ளி விவரங்களுக்கு பின்வரும் கணிப்பான் *h(x)* மூலம் கணிப்பதை நாம் இங்கு உதாரணமாக எடுத்துக் கொள்வோம்.

$$h(x) = \theta_0 + \theta_1 x$$

இந்தக் கணிப்பானது தீட்டா-0 மற்றும் தீட்டா-1 எனும் இரண்டு முக்கிய *parameters*-ஐப் பொறுத்தே அமைகிறது. இவற்றையே முன்னர் *alpha, beta* என அழைத்தோம். வெவ்வேறு மதிப்புள்ள *parameters*-க்கு வெவ்வேறு வகையில் கணிப்புகள் நிகழ்த்தப்படுவதை பின்வரும் உதாரணத்தில் காணலாம்.

https://gist.github.com/nithyadurai87/c57ac1197368249f015ed4d1dba029f0

```
import matplotlib.pyplot as plt

x = [1, 2, 3]
y = [1, 2, 3]

plt.figure()
plt.title('Data - X and Y')
plt.plot(x,y,'*')
plt.xticks([0,1,2,3])
```

```
plt.yticks([0,1,2,3])
plt.show()

def linear_regression(theta0,theta1):
	predicted_y = []
	for i in x:
		predicted_y.append((theta0+(theta1*i)))
	plt.figure()
	plt.title('Predictions')
	plt.plot(x,predicted_y,'.')
	plt.xticks([0,1,2,3])
	plt.yticks([0,1,2,3])
	plt.show()

theta0 = 1.5
theta1 = 0
linear_regression(theta0,theta1)

theta0a = 0
theta1a = 1.5
linear_regression(theta0a,theta1a)

theta0b = 1
theta1b = 0.5
linear_regression(theta0b,theta1b)
```

முதலில் (1,1) , (2,2) , (3,3) -க்கான வரைபடம் வரையப்படுகிறது. பின்னர் தீட்டா-0 =1.5, தீட்டா-1 =0 எனும் போது பின்வரும் சமன்பாட்டில் பொருத்தி (1, 1.5) , (2, 1.5) , (3, 1.5) எனும் மதிப்புகளையும்,

h(1) = 1.5 + 0(1) = 1.5

h(2) = 1.5 + 0(2) = 1.5

h(3) = 1.5 + 0(3) = 1.5

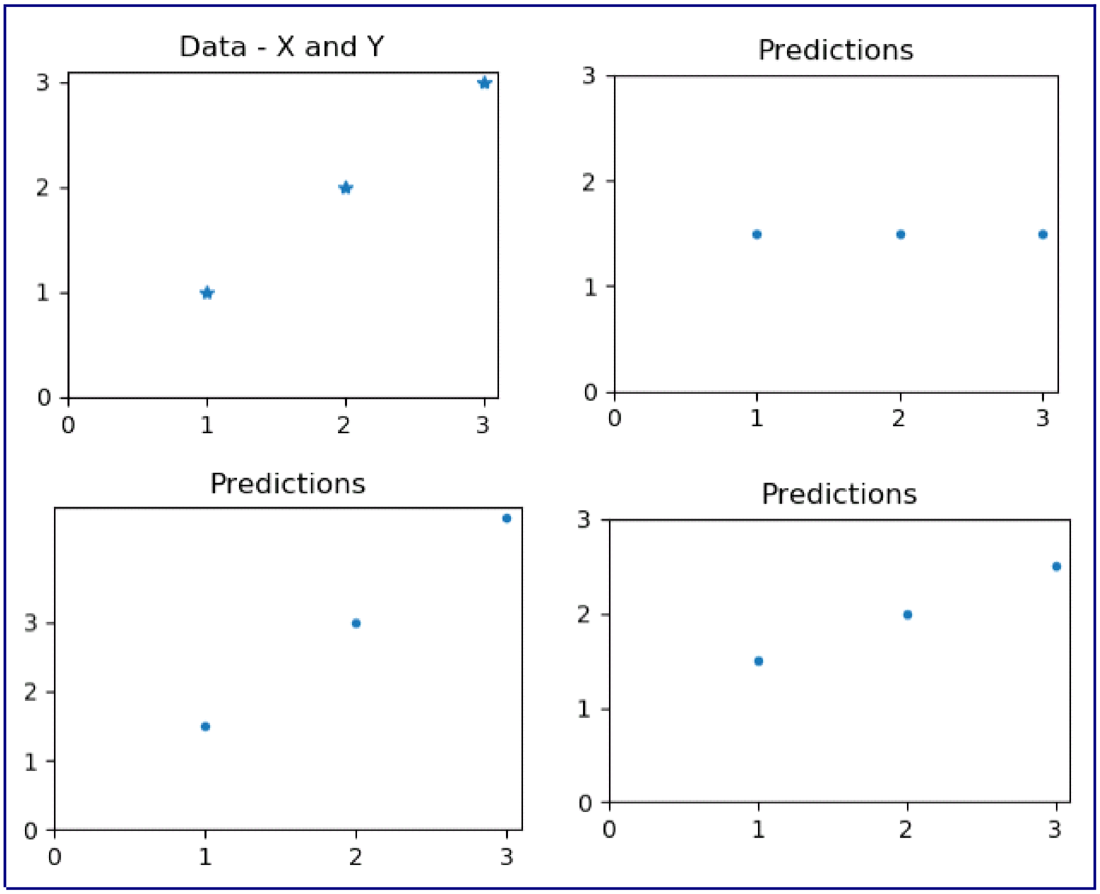
அவ்வாறே தீட்டா-0 =0, தீட்டா-1 =1.5 எனும் போது (1, 1.5) , (2, 3) , (3, 4.5) எனும் மதிப்புகளையும், கடைசியாக தீட்டா-0 =1, தீட்டா-1 =0.5 எனும் போது (1, 1.5) , (2, 2) , (3, 2.5) எனும் மதிப்புகளையும் அளிப்பதைக் காணலாம்.

h(1) = 0 + 1.5(1) = 1.5 h(1) = 1 + 0.5(1) = 1.5

h(2) = 0 + 1.5(2) = 3 h(2) = 1 + 0.5(2) = 2

h(3) = 0 + 1.5(3) = 4.5 h(3) = 1 + 0.5(3) = 2.5

இவ்வாறு கண்டுபிடிக்கப்பட்ட மதிப்புகளுக்கு வரைபடங்கள் வரையப்படுகின்றன. இவை பின்வருமாறு அமையும்.



மேற்கண்ட 3 கணிப்புகளில் எதன் கணிப்பு உண்மையான மதிப்புகளுக்கு அருகில் உள்ளதோ அதனுடைய தீட்டா மதிப்புகளையே நாம் இறுதியாக கணிப்பிற்கு எடுத்துக்கொள்ளலாம். இங்கு (1,1) , (2,2) , (3,3) எனும் மதிப்புகளுக்கு (1, 1.5) , (2, 2) , (3, 2.5) எனும் மதிப்புகள் சற்று அருகில் வந்துள்ளது. எனவே தீட்டா-0 =1, தீட்டா-1 =0.5 எனும் மதிப்புகளைக் கொண்ட கணிப்பானையே நாம் தேர்வு செய்து கொள்வோம்.

இங்கு வெறும் 3 தரவுகள் மட்டும் இருப்பதால், எதை வைத்துக் கணித்தால் கணிப்பிற்கும் உண்மையான மதிப்பிற்குமான வேறுபாடு சற்று குறைவாக இருக்கும் என நம்மால் சுலபமாகக் கூற முடியும். ஆனால் நிஜத்தில் ஆயிரக்கணக்கில் தரவுகள் இருக்கும்போது, இந்த வேறுபாட்டினைக் கண்டறிந்து கூற உதவும் சூத்திரமே cost function ஆகும்.

Cost Function: இதற்கான சமன்பாடு பின்வருமாறு.

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

J = cost function

m = மொத்த தரவுகளின் எண்ணிக்கை

i = மொத்தத் தரவுகளில் ஒவ்வொன்றாகச் செல்ல உதவும். .

h(x) = கணிக்கப்படுகின்ற மதிப்பு

y = எதிர்பார்க்கின்ற உண்மையான மதிப்பு

மேற்கண்ட அதே புள்ளி விவரங்களை பின்வரும் சமன்பாட்டில் பொருத்தி, நாம் தேர்ந்தெடுத்துள்ள கணிப்பான் சிறிய அளவு cost வேறுபாட்டினை வெளிப்படுத்துகிறதா எனக் காணவும். இதற்கான நிரல் பின்வருமாறு.

https://gist.github.com/nithyadurai87/86bd4ec2288d0e9af138a30a7af44a09

<u>வெளியீடு:</u>

cost when theta0=1.5 theta1=0 : 0.4583333333333333
cost when theta0=0 theta1=1.5 : 0.5833333333333333
cost when theta0=1 theta1=0.5 : 0.08333333333333333

<u>கணக்கீடு நிகழும் விதம்:</u>

(1, 1.5) , (2, 1.5) , (3, 1.5)　　vs　　(1, 1) , (2, 2) , (3, 3)　　(தீட்டா-0 =1.5, தீட்டா-1 =0)

*J = 1/2*3 [(1.5-1)**2 + (1.5-2)**2 + (1.5-3)**2] = 1/6 [0.25 + 0.25 + 2.25] = 2.75*

(1, 1.5) , (2, 3) , (3, 4.5)　　vs　　(1, 1) , (2, 2) , (3, 3)　　(தீட்டா-0 =0, தீட்டா-1 =1.5)

*J = 1/2*3 [(1.5-1)**2 + (3-2)**2 + (4.5-3)**2] = 1/6 [0.25 + 1 + 2.25] = 3.50*

(1, 1.5) , (2, 2) , (3, 2.5)　　vs　　(1, 1) , (2, 2) , (3, 3)　　(தீட்டா-0 =1, தீட்டா-1 =0.5)

*J = 1/2*3 [(1.5-1)**2 + (2-2)**2 + (2.5-3)**2] = 1/6 [0.25 + 0 + 0.25] = 0.50*

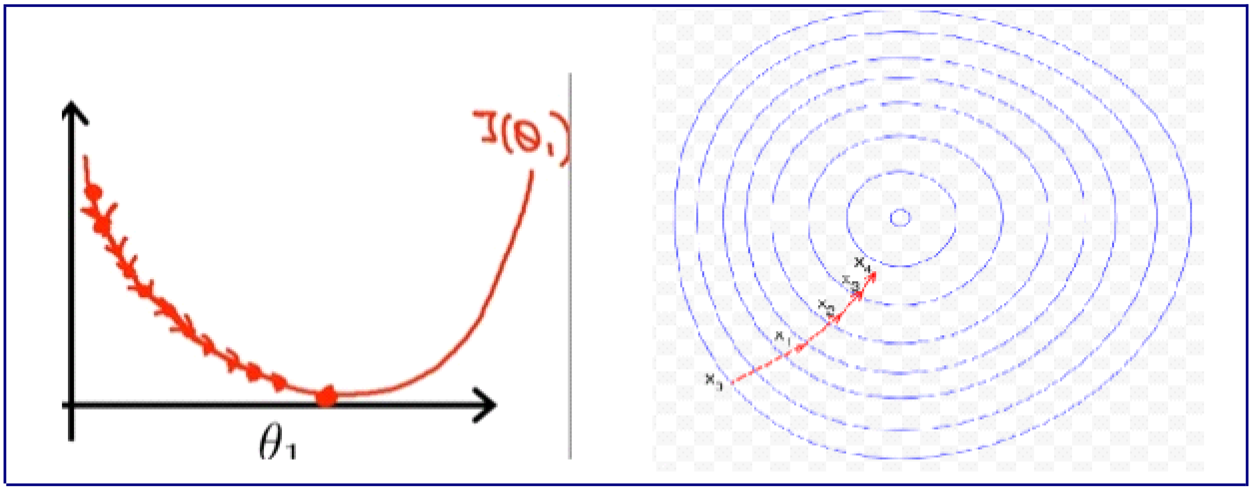
இதில் தனித்தனி வேறுபாடுகளைக் கூட்டி அதன் சராசரியைக் கண்டுபிடிப்பதன் மூலம் ஒவ்வொரு கணிப்பான் நடத்தும் கணிப்பும் எந்த அளவு வேறுபாடில் அமையும் என்பதைக் கூற முடியும்.இத்தகைய வேறுபாடுகளின் மடங்குகள் கண்டுபிடிக்கப்பட்டு அவை 2-ஆல் வகுக்கப்படுவதற்கான காரணம் என்னவெனில், சராசரியைக் கண்டுபிடிக்கும்போது ஏதாவது ஒரு எதிர்மறை எண் இருந்தால்கூட அது கூட்டப்படுவதற்கு பதிலாக கழிக்கப்பட்டு விடும். எனவேதான் சூத்திரம் இதுபோன்று அமைக்கப்பட்டுள்ளது. இதுவே sum of squares error என்று அழைக்கப்படும்.

இங்கு நாம் ஏற்கனவே கண்டுபிடித்த தீட்டா-0 =1, தீட்டா-1 =0.5 மதிப்புகள் கொண்ட கணிப்பானே குறைந்த அளவு cost-ஐ வெளிப்படுத்துவதைக் காணலாம்(0.50). எனவே தரவுகளின் எண்ணிக்கை பெருகினாலும், இந்த சூத்திரத்தின் மூலம் வேறுபாட்டினை நாம் சுலபமாகக் கணக்கிடாலாம்.

இந்த 0.50 எனும் வேறுபாடு சற்று அதிகம் எனக் கருதினால், இதனை நாம் இன்னும் குறைக்க பல்வேறு தீட்டாக்களுக்கு இச்சோதனையை திரும்பத் திரும்ப செய்து அதில் குறைவான வேறுபாடு ஏற்படுத்தும் தீட்டாக்களை கண்டுபிடிக்க உதவுவதே Gradient descent ஆகும். அதற்கு முதலில் பல்வேறு தீட்டாக்களின் மதிப்பையும், அவற்றுக்கான cost-ஐயும் ஒரு வரைபடமாக வரைந்து பார்ப்போம். இதுவே contour plots ஆகும்.

<u>Contour plots:</u>

தீட்டா-0 , தீட்டா-1 மற்றும் இவ்விரண்டின் அடிப்படையில் கண்டறியப்பட்ட cost மதிப்பு இம்மூன்றையும் முப்பரிமாண வரைபடமாக வரைந்து காட்ட உதவுவதே contour வரைபடம் ஆகும். இது கிண்ண வடிவிலோ அல்லது வட்ட வடிவிலோ பின்வருமாறு இருக்கும். புள்ளி வைத்துள்ள இடங்களில் எல்லாம் cost இருக்கிறது என வைத்துக் கொண்டால், அவற்றை எல்லாம் இணைப்பதன் மூலம் கிண்ணம் போன்ற ஒரு வடிவம் ஏற்படும்.



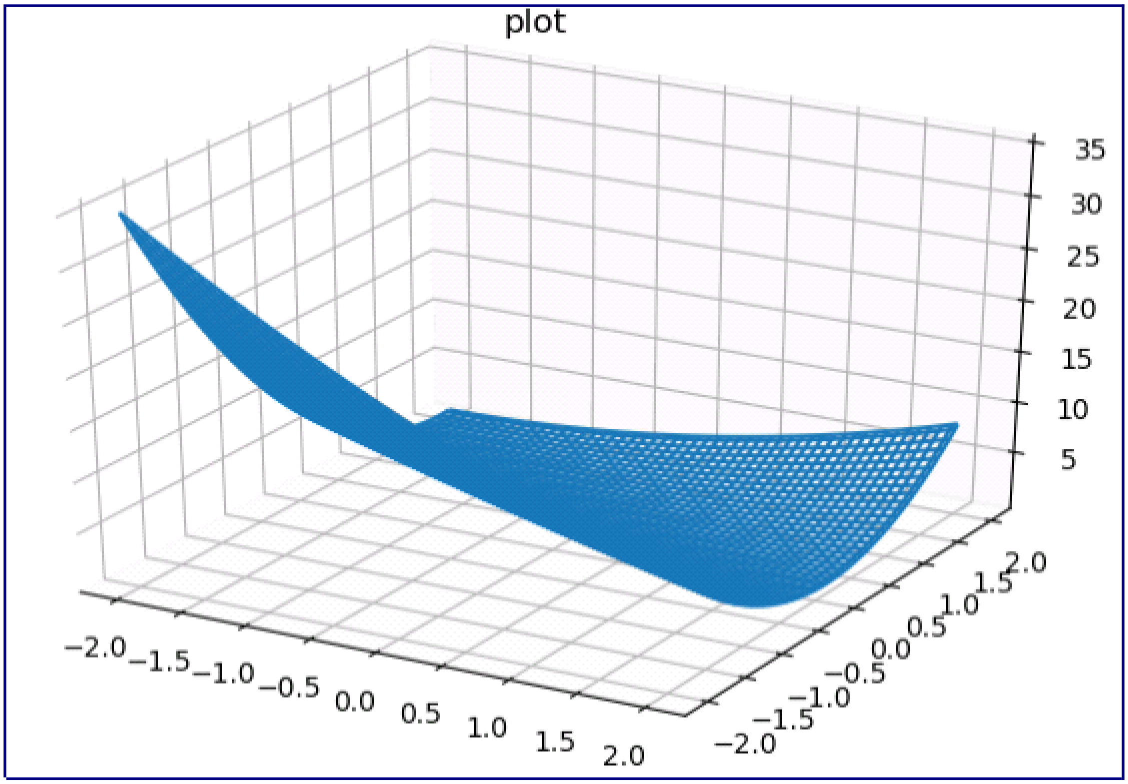
வட்டம் போன்ற வரைபடத்தில் பல்வேறு தீட்டா மதிப்புகளுக்கான *cost* பல்வேறு வட்டங்களாக வெளிப்படுகின்றன. எனவே வட்டத்தின் மையத்தைக் கண்டறிவதன் மூலமோ அல்லது கிண்ணத்தின் அடிப்பாகத்தை அடைவதன் மூலமோ குறைந்த அளவு வேறுபாட்டினை வெளிப்படுத்தக் கூடிய தீட்டாக்களை நாம் கண்டறிய முடியும். இந்த வேலையையே *Gradient descent* செய்கிறது.

கீழ்க்கண்ட நிரலில் *-2* லிருந்து *2* வரை தீட்டாக்களுக்கு *100* முறை மதிப்புகள் மாற்றி மாற்றி அளிக்கப்பட்டு *cost* கண்டறியப்படுகிறது. இங்கு *numpy* மூலம் தீட்டாக்களுக்கு மதிப்புகள் வழங்கப்படுகின்றன. இவை *uniform distribution* முறையில் அமையும்.

https://gist.github.com/nithyadurai87/8c120370181f5bb9ad966dc9fdd7935b

```
from mpl_toolkits.mplot3d.axes3d import Axes3D
import matplotlib.pyplot as plt
import numpy as np

fig, ax1 = plt.subplots(figsize=(8, 5),
                        subplot_kw={'projection': '3d'})


values = 2
r = np.linspace(-values,values,100)

theta0,theta1= np.meshgrid(r,r)

original_y = [1, 2, 3]
m = len(original_y)

predicted_y = [theta0+(theta1*1), theta0+(theta1*2), theta0+(theta1*3)]

sum=0
for i,j in zip(predicted_y,original_y):
  sum = sum+((i-j)**2)
J = 1/(2*m)*sum

ax1.plot_wireframe(theta0,theta1,J)
ax1.set_title("plot")

plt.show()
```

நமது தரவுகளுக்கான வரைபடம் பின்வருமாறு.



Gradient descent

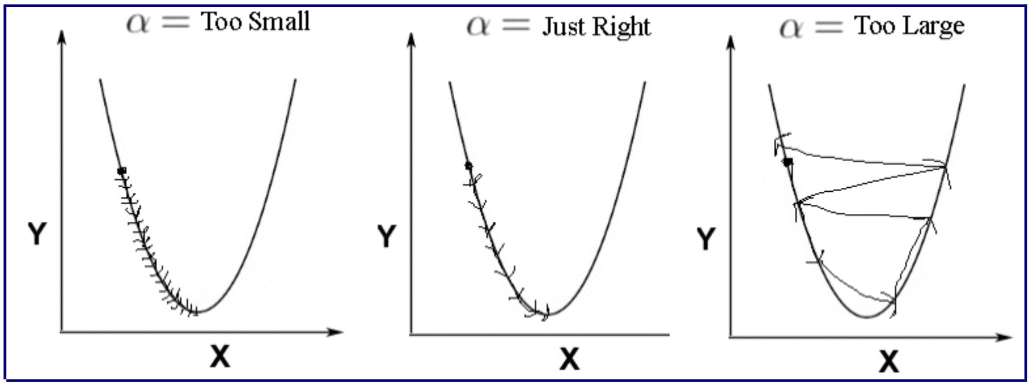
குறைந்த அளவு வேறுபாடு ஏற்படுத்தக் கூடிய தீட்டாக்களின் மதிப்பினைக் கண்டுபிடிக்கும் வேலையை *gradient descent* செய்கிறது முதலில் தீட்டாக்களுக்கு ஒரு குறிப்பிட்ட மதிப்பினைக் கொடுத்து அதற்கான *cost*-ஐக் கண்டறிகிறது. பின்னர் அம்மதிப்பிலிருந்து, ஒரு குறிப்பிட்ட அளவு விகிதத்தில் தீட்டாக்களின் மதிப்புகள் குறைக்கப்பட்டு அதற்கான *cost* கண்டறியப்படுகிறது. இவ்வாறாக ஒவ்வொரு சுழற்சியிலும் சிறிது சிறிதாகக் குறைத்துக் கொண்டே வந்து குறைந்த அளவு *cost* கண்டுபிடிக்கப்படுகிறது. . இதற்கான சமன்பாடு பின்வருமாறு.

$$\theta_0 := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J$$
$$\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J$$

இங்கு ஒவ்வொரு சுழற்சியின் முடிவிலும் தீட்டா-0 , தீட்டா-1 ஆகியவற்றின் மதிப்புகள் ஒரே நேரத்தில் குறைக்கப்பட வேண்டும். இதுவே *simultaneous update* எனப்படுகிறது. கிண்ணமாக இருப்பின், கிண்ணத்தின் அடிப்பகுதியைக் கண்டறிவதும், வட்டமாக இருப்பின் அவ்வட்டத்தின் மையத்தைக் கண்டறியும் வேலையையுமே இந்த *gradient descent* செய்கிறது. இதுவே *global optimum*-ஐ அடையும் வழி ஆகும். இதற்கு மாறாக *local optimum* என்பது *cost*-ன் மதிப்புகளில் தொடர்ச்சியாக ஏற்ற இறக்கங்கள் இருப்பின், ஒவ்வொரு இறக்கமும் *local optimum* எனப்படும். பொதுவாக *linear regression*-

க்கான வரைபடத்தில், local minimum என்பது கிடையாது. global optimum மட்டுமே.

Alpha என்பது தீட்டாக்களின் மதிப்புகள் எந்த அளவு விகிதத்தில் குறைக்கப்பட வேண்டும் என்பதைக் குறிக்கும். இதன் மதிப்பு மிகவும் சிறியதாகவும் இல்லாமல், மிகவும் பெரியதாகவும் இல்லாமல் சரியான அளவில் அமைய வேண்டும். பொதுவாக 0.1, 0.01, 0.001 என்று அமையும்.



மேற்கண்ட மூன்று படங்களிலும் J-ன் மதிப்பு புள்ளி வைத்துள்ள இடத்தில் இருக்கிறது என வைத்துக்கொள்வோம். இப்போது alpha-ன் மதிப்பு மிகச்சிறியதாக இருந்தால், புள்ளி வைத்துள்ள இடத்திலிருந்து ஒவ்வொரு சுழற்சிக்கும் சிறிது சிறிதாகக் குறைக்கப்பட்டு global optimum-ஐ அடைவதற்கு மிகுந்த நேரம் பிடிக்கும். அதிக அளவு சுழற்சிகளும் தேவைப்படும். ஒரு குழந்தை சின்னச் சின்ன அடியாக அடியெடுத்து வைப்பது போல இது நகரும். அதுவே மிகவும் அதிகமாக இருந்தால்கூட, J-ன் மதிப்பு global optimum-க்கு மிக அருகாமையில் வந்தாலும் கூட, அடுத்த அடியை மிக நீளமாக எடுத்து வைப்பதால், global optimum-ஐயும் தாண்டி, வேறு எங்கோ சென்று விடும். அவ்வாறே அடுத்தடுத்த சுழற்சிகளில் எங்கெங்கோ சென்று மையத்தினைச் சென்றடையத் தவரும். எனவே alpha-ன் மதிப்பினை சரியான அளவில் கொடுக்க வேண்டும்.

Alpha-க்கு அடுத்து ஒவ்வொரு தீட்டாவின் partial derivative கண்டுபிடிக்கப்படுகிறது. Simple linear regression-ல் தீட்டா-0 என்பது தனியாக இருக்கும். தீட்டா-1 என்பது x-வுடன் சேர்ந்து இருக்கும்(h(x) = தீட்டா-0 + தீட்டா-1x). partial derivative -க்கான சமன்பாடும் இதே முறையில் பின்வருமாறு அமையும்.

$$\frac{\partial}{\partial\theta_0} J = \frac{1}{m}\sum_{i=1}^{m}\left(h(x) - y\right) \qquad \text{(for } \theta_0 \text{)}$$

$$\frac{1}{m}\sum_{i=1}^{m}\left(h(x) - y\right)\cdot x \qquad \text{(for } \theta_1 \text{)}$$

கீழ்க்கண்ட நிரலில் தீட்டா0 மதிப்பு நிலையாக வைக்கப்பட்டு, தீட்டா1 -ன் மதிப்பு மட்டும் gradient descent முறையில் குறைக்கப்படுகிறது. 50 சுழற்சிகள் நடத்தப்படுகின்றன. பின்னர் J_history எனும் list-ல் ஒவ்வொரு சுழற்சியிலும் கண்டறியப்பட்ட cost சேமிக்கப்பட்டு அதிலிருந்து குறைவான மதிப்பு தேர்வு செய்யப்படுகிறது. இதில் partial derivative-ஆனது delta எனும் பதத்தால் குறிக்கப்படுகிறது. இதன் மதிப்பு பின்வரும் வகையில் கணக்கிடப்படுகிறது.

delta = 1/m . (h(x) - y).x

= 1/m . (தீட்டா1.x – y).x where h(x)= தீட்டா1.x

= 1/m . x. தீட்டா1.x – x.y

https://gist.github.com/nithyadurai87/43664cacd625e7c290c8812894dca659

```
x = [1, 2, 3]
y = [1, 2, 3]

m = len(y)

theta0 = 1
theta1 = 1.5
alpha = 0.01

def cost_function(theta0,theta1):
	predicted_y = [theta0+(theta1*1), theta0+(theta1*2), theta0+(theta1*3)]
	sum=0
	for i,j in zip(predicted_y,y):
		sum = sum+((i-j)**2)
	J = 1/(2*m)*sum
	return (J)

def gradientDescent(x, y, theta1, alpha):
	J_history = []
	for i in range(50):
		for i,j in zip(x,y):
			delta=1/m*(i*i*theta1-i*j);
			theta1=theta1-alpha*delta;
			J_history.append(cost_function(theta0,theta1))
	print (min(J_history))


gradientDescent(x, y, theta1, alpha)
```

வெளியீடு:

0.5995100694321308

Gradient descent-ல் அதிகபட்ச சுழற்சிகள் எவ்வளவு இருக்க வேண்டும் எனக் கொடுத்து, அதிலிருந்து குறைவான cost-ஐ தேர்வு செய்யலாம். மேலும் அடுத்தடுத்த தொடர்ச்சியான சுழற்சிகளில் ஒரே மாதிரியான cost மதிப்புகள் வெளிப்படுகிறதெனில் நாம் global optimum-ஐ அடைந்து விட்டோம் என்று அர்த்தம். உதாரணத்துக்கு 300 முதல் 400 வரையிலான சுழற்சிகளில் J மதிப்பு, மிக மிகக் குறைந்த அளவே வேறுபடுகிறதெனில் (<0.001) அது global optimum-ஐ அடைந்து விட்டது என்றே அர்த்தம். இதுவே Automatic convergence test என்றும் அழைக்கப்படுகிறது.

Matrix

பல்வேறு எண்கள் அணிவகுத்துச் செல்வது அணிகள் எனப்படும். simple linear regression-ல் ஒரே ஒரு எண்ணை வைத்துக் கொண்டு வேறொரு எண்ணைக் கணித்தோம். ஆனால் இனிவரும் multiple linear-ல் ஒன்றுக்கும் மேற்பட்ட எண்கள் ஒன்றாகச் சேர்ந்து வேறொரு எண்ணைக் கணிக்கப் போகிறது. அதாவது ஒரு வீட்டின் சதுர அடி விவரத்தை மட்டும் வைத்துக் கொண்டு, அவ்வீட்டின் விலையைக் கணிப்பது simple linear எனில், ஒரு வீட்டின் சதுரஅடி, அறைகளின் எண்ணிக்கை, எத்தனை வருடம் பழையது போன்ற பல்வேறு காரணிகளை வைத்துக்கொண்டு அவ்வீட்டின் விலையைக் கணிப்பது multiple linear ஆகும். எனவே அதைப் பற்றிக் கற்பதற்கு முன்னர் ஒன்றுக்கும் மேற்பட்ட எண்களை எவ்வாறு அணிவகுப்பது, அணி வகுக்கப்பட்ட எண்களை வைத்து எவ்வாறு கணக்கீடுகள் செய்வது போன்ற ஒரு சில அடிப்படை விஷயங்களைக் கற்றுக் கொள்ள வேண்டும்.

அணி:

*ஒரு அணியில் எத்தனை rows மற்றும் columns உள்ளது என்பதே அந்த அணியின் dimension எனப்படும். 2 rows மற்றும் 3 columns கொண்ட A அணி பின்வருமாறு அமையும். இது 2 * 3 dimensional matrix எனப்படும். இந்த அணியில் உள்ள மதிப்புகளை அணுக, A -ன் கீழ் எத்தனையாவது row மற்றும் எத்தனையாவது column-ல் அம்மதிப்பு உள்ளது எனக் கொடுக்க வேண்டும். உதாரணத்துக்கு A22 என்பது இரண்டாவது row மற்றும் இரண்டாவது column-ல் உள்ள 5 எனும் மதிப்பினைக் குறிக்கும்.*

Matrix

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \quad (2 \times 3) matrix.$$

$$\mathrm{A}_{22} = 5$$

Multiple linear என்று வரும்போது ஒரு அணியின் dimension என்பது தரவுகளின் எண்ணிக்கை மற்றும் அது கணிப்பதற்கு எடுத்துக் கொள்ளும் அம்சங்கள் ஆகியவற்றைப் பொறுத்து அமையும். அதாவது.

rows = no. of records

columns = no. of features

வெக்டர்:

ஒரே ஒரு column-ஐக் கொண்ட அணி வெக்டர் என்று அழைக்கப்படும். இது பின்வருமாறு. வெக்டரில் உள்ள மதிப்புகளை அணுக எத்தனையாவது row என்று மட்டும் கொடுத்தால் போதுமானது. B3 என்பது மூன்றாவது row-ல் உள்ள மதிப்பான 38 என்பதைக் குறிக்கும். ஒரு வெக்டரை 0-indexed மற்றும் 1-indexed எனும் இரு வகைகளில் குறிக்கலாம். B3 என்பது 1-indexed எனில் 38-ஐயும், 0-indexed எனில் 47-ஐயும் குறிக்கும்.

Vector

$$B = \begin{bmatrix} 15 \\ 20 \\ 38 \\ 47 \end{bmatrix} \quad 4 - dimensionvector$$

$$\mathrm{B}_3 = 38 \quad (1 - indexed)$$

அணிகளின் கூட்டல்:

இரண்டு அணிகளின் dimension சமமாக இருந்தால் மட்டுமே அவ்விரண்டு அணிகளையும் கூட்டி அதே dimension கொண்ட மற்றொரு அணியை உருவாக்க முடியும்.

*If (3*2) * (3*2) = 3*2*

1+7 8+2

3+9 4+10

5+11 6+12

Matrix Addition

$$\begin{bmatrix}1 & 2\\3 & 4\\5 & 6\end{bmatrix}+\begin{bmatrix}7 & 8\\9 & 10\\11 & 12\end{bmatrix}=\begin{bmatrix}8 & 10\\12 & 14\\16 & 18\end{bmatrix}$$

$$(3\times 2) \qquad (3\times 2) \qquad (3\times 2)$$

<u>அணிகளின் பெருக்கல்:</u>

முதலாவது அணியின் column மற்றும் இரண்டாவது அணியின் row ஆகியவைகளின் எண்ணிக்கை சமமாக இருந்தால் மட்டுமே அவ்விரண்டு அணிகளையும் பெருக்கி மற்றொரு அணியை உருவாக்க முடியும். புதிதாக பெருக்கி உருவாக்கப்பட்ட அணியின் dimension-ஆனது, முதலாவது அணியின் rows மற்றும் இரண்டாவது அணியின் columns மதிப்பினைப் பெற்றிருக்கும்.

*If (3*2) * (2*2) = 3*2*

முதலாவது அணியில் உள்ள row-ன் மதிப்புகள் இரண்டாவது அணியில் உள்ள column-ன் மதிப்புகளுடன் தனித்தனியாகப் பெருக்கப்படும். பின்னர் அப்பெருக்களின் மதிப்புகள் ஒன்றாகக் கூட்டப்படுகின்றன. இவ்வாறே அணிகளின் பெருக்கல் நடைபெறுகிறது.

*(1*7)+(2*9)*	*(1*8)+(2*10)*	=	*7+18*	*8+20*
*(3*7)+(4*9)*	*(3*8)+(4*10)*	=	*21+36*	*24+40*
*(5*7)+(6*9)*	*(5*8)+(6*10)*	=	*35+54*	*40+60*

Matrix Multiplication

$$\begin{bmatrix}1 & 2\\3 & 4\\5 & 6\end{bmatrix}\times\begin{bmatrix}7 & 8\\9 & 10\end{bmatrix}=\begin{bmatrix}7+18 & 8+20\\21+36 & 24+40\\35+54 & 40+60\end{bmatrix}=\begin{bmatrix}25 & 28\\57 & 64\\89 & 100\end{bmatrix}$$

$$(3\times 2) \qquad (2\times 2) \qquad\qquad (3\times 2)$$

<u>அணிகளின் transpose:</u>

ஒரு அணியில் உள்ள rows அனைத்தும் columns-ஆக மாற்றப்படுவதே அந்த அணியின் transpose எனப்படும்.

Matrix Transpose

$$A=\begin{bmatrix}1 & 2\\3 & 4\\5 & 6\end{bmatrix}\quad A^T=\begin{bmatrix}1 & 3 & 5\\2 & 4 & 6\end{bmatrix}$$

<u>அணிகளின் Inverse:</u>

*ஒரு அணியின் inverse என்பது சற்றே கடினமான முறையில் கணக்கிடப்படும். 2*2 dimension கொண்ட அணியின் inverse பின்வருமாறு கணக்கிடப்படும். A11, A22 மதிப்புகளின் பெருக்கலுக்கும் A12, A21 மதிப்புகளின் பெருக்கலுக்கும் உள்ள வித்தியாசமானது 1-ன் கீழ் அமைந்து வகுக்கப்படும். இதன்*

தொடர்ச்சியாக அதே அணியில் உள்ள A11, A22 மதிப்புகள் இடமாற்றம் செய்யப்பட்டும், A12, A21 மதிப்புகள் எதிர் மறையில் மாற்றப்பட்டும் பெருக்கப்படும்.

Matrix Inverse

$$A = \begin{bmatrix} 2 & 4 \\ 3 & 7 \end{bmatrix}$$

$$A^{-1} = \frac{1}{14 - 12} \begin{bmatrix} 7 & -4 \\ -3 & 2 \end{bmatrix} = \begin{bmatrix} 3.5 & -2 \\ -1.5 & 1 \end{bmatrix}$$

Identity Matrix:

ஒரே எண்ணிக்கையிலான rows மற்றும் columns-ஐக் கொண்ட அணியே சதுர அணி எனப்படும். ஒரு சதுர அணியின் மூலைவிட்டத்தில் மட்டும் 1 என இருந்து மற்ற இடங்களில் எல்லாம் பூஜ்ஜியம் என இருந்தால் அதுவே Identity Matrix எனப்படும். ஒரு அணியும், அந்த அணியின் inverse-ம் சேர்ந்து Identity matrix-ஐ உருவாக்கும்.

Identity Matrix

$$I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$A(A^{-1}) = I$$

Multiple Linear Algorithm

ஒன்றுக்கும் மேற்பட்ட அம்சங்கள் ஒன்றாகச் சேர்ந்து ஒரு விஷயத்தைக் கணிக்கிறது எனில் அதுவே multiple linear regression எனப்படும். ஒவ்வொரு அம்சமும் x1,x2,x3.. எனக் கொண்டால், இதற்கான சமன்பாடு பின்வருமாறு அமையும்.

$$\begin{aligned} h(x) &= \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n \\ &= \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n \quad \textbf{(added } x_0 = 1\textbf{)} \\ &= \theta^T x \end{aligned}$$

multiple linear-ல் ஒவ்வொரு feature-க்கும் ஒரு தீட்டா மதிப்பு காணப்படுமே தவிர, no.of rows -ஐப்

பொறுத்து மாறாது. எனவே தீட்டா என்பது எப்போதும் 1 row-ல் பல்வேறு மதிப்புகள் அமைந்துள்ள அணியாக இருக்கும். பின்னர் இந்த அணியை transpose செய்து 1 column-ல் பல்வேறு மதிப்புகள் அமைந்துள்ள வெக்டராக மாற்றலாம். எனவே தான் transpose செய்யப்பட்ட தீட்டா அணியையும், features-க்கான X அணியையும் பெருக்கினால் multiple linear-க்கான சமன்பாடு வந்துவிடுகிறது.

இந்த சமன்பாட்டில் தீட்டா0 -வுடன் x0 எனும் புதிய feature ஒன்று சேர்க்கப்படுகிறது. இது எப்போதும் 1 எனும் மதிப்பையே பெற்றிருக்கும். இந்த புதிய feature-ஆல் தீட்டா0 மதிப்பில் எந்த ஒரு மாற்றமும் ஏற்படாது. வெறும் அணிகளின் பெருக்கலுக்கு துணைபுரியும் வகையில் இது சேர்க்கப்பட்டுள்ளது.

கீழ்க்கண்ட உதாரணத்தில்,

800 சதுர அடி, 2 அறைகள், 15 வருட பழைய வீட்டின் விலை = 3000000

1200 சதுர அடி, 3 அறைகள், 1 வருட பழைய வீட்டின் விலை = 2000000

2400 சதுர அடி, 5 அறைகள், 5 வருட பழைய வீட்டின் விலை = 3500000

எனும் 3 தரவுகள் X எனும் அணியில் கொடுக்கப்பட்டுள்ளன. அவ்வாறே 100, 1000, 10000, 100000 ஆகிய மதிப்புகள் தீட்டா0 , தீட்டா1, தீட்டா2, தீட்டா3 -ன் மதிப்புகளாக தீட்டா எனும் அணியில் கொடுக்கப்பட்டுள்ளன. இவை இரண்டும் மேற்கண்ட சமன்பாட்டின் படி பொருத்தப்பட்டு, h(x) அணியை உருவாக்குகின்றன.

https://gist.github.com/nithyadurai87/5abf51e4b26717a3427d15fcaca6f48f

```
import matplotlib.pyplot as plt
import numpy as np

x = np.array([[1, 800, 2, 15],[1, 1200, 3, 1],[1, 2400, 5, 5]])
y = np.array([3000000,2000000,3500000])
theta = np.array([100, 1000, 10000, 100000])

predicted_y = x.dot(theta.transpose())
print (predicted_y)

m = y.size
diff = predicted_y - y
squares = np.square(diff)
#sum_of_squares = 5424168464
sum_of_squares = np.sum(squares)
cost_fn = 1/(2*m)*sum_of_squares
print (diff)
print (squares)
print (sum_of_squares)
print (cost_fn)
```

வெளியீடு:

[2320100 1330100 2950100]
[-679900 -669900 -549900]
[462264010000 448766010000 302390010000]

1213420030000

202236671666.66666

கணக்கிடு நிகழ்ந்த விதம்:

$$\begin{bmatrix} 1 & 800 & 2 & 15 \\ 1 & 1200 & 3 & 1 \\ 1 & 2400 & 5 & 5 \end{bmatrix} * \begin{bmatrix} 100 \\ 1000 \\ 10000 \\ 100000 \end{bmatrix} = \begin{bmatrix} 2320100 \\ 1330100 \\ 2950100 \end{bmatrix}$$

<u>Cost function:</u>

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

இது *simple linear*-ஐ ஒத்தே இருக்கும். ஆனால் *h(x)* கணக்கிடும் சூத்திரம் மட்டும் மாறுபடும்.

<u>Gradient descent:</u>

இதுவும் *simple linear*-ஐ ஒத்தே இருக்கும். ஆனால் *simple linear*-ல் தீட்டா*0* மதிப்பு குறைக்கப்படுவதற்கான சமன்பாடில் *x* என்பது இருக்காது. ஆனால் இங்கு தீட்டா*0* -வுடன் *x0* சேர்க்கப்பட்டிருப்பதால், அனைத்து தீட்டா மதிப்புகள் குறைக்கப்படுவதற்கான சமன்பாடும் பின்வருமாறு ஒரே மாதிரியாகத்தான் இருக்கும்.

$$\theta_0 := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J$$

$$\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J$$

where as,

$$\frac{\partial}{\partial \theta_0} J = \frac{1}{m} \sum_{i=1}^{m} \left(h(x) - y \right) \cdot x \quad \textbf{(for all } \theta \textbf{)}$$

<u>சூத்திரத்தின் மூலம் minimum cost கண்டறிதல்:</u>

Gradient descent-ஐப் பயன்படுத்துவதற்கு பதிலாக பின்வரும் சமன்பாட்டின் மூலம் நேரடியாக நாம் *minimum cost*-ஐ ஏற்படுத்தக் கூடிய தீட்டாவை கண்டுபிடிக்க முடியும். ஆனால் *features*-ன் எண்ணிக்கை அதிகமாக இருந்தால், *gradient descent*-ஐப் பயன்படுத்துவதே சிறந்தது. ஏனெனில் மொத்த *features*-க்கும் அதனுடைய *transpose* கண்டுபிடிப்பது மிகுந்த நேர விரயம் செய்யக்கூடியதாக அமையும்.

$$\theta = (X^T X)^{-1} X^T y$$

<u>Feature Scaling:</u>

இங்கு ஒவ்வொரு *feature*-ம் வெவ்வேறு அளவிலான எண் வரிசைகளில் அமைந்திருப்பதைக் கவனிக்கவும். உதாரணத்துக்கு சதுர அடி என எடுத்துக் கொண்டால் அவை *800* முதல் *1200* வரையிலும், அறைகளின் எண்ணிக்கை என எடுத்துக்கொண்டால் அவை *2* முதல் *5* வரையிலும் பரவியுள்ளது.

சதுர அடி = *800, 1200, 2400*

அறைகள் = 2, 3, 5

இவ்வாறு ஒவ்வொரு *column*-ல் உள்ள மதிப்புகளும் வெவ்வேறு எண் வரிசைகளில் இல்லாமல், அனைத்தும் *-1* லிருந்து *+1* வரை அல்லது *0* லிருந்து *1* வரை என *normalize* செய்வதே *feature scaling* எனப்படும். இதற்கு உதவுவதே *mean normalization* ஆகும். இதற்கான சூத்திரம் பின்வருமாறு.

particular value – mean of all values

\---

maximum – minimum

சதுர அடி = (800 –1600)/(2400–800), (1200–1600)/(2400–800), (2400–1600)/(2400–800)

= -0.5 , 0.25, 0.5

அறைகள் = (2-3.5)/(5–2), (3-3.5)/(5–2), (5-3.5)/(5–2)

= -0.5 , -0.16, 0.5

இது போன்ற *multiple linear*-ல் *gradient descent*-ஐப் பயன்படுத்தும்போது ஒவ்வொரு *feature*-ம் ஒவ்வொரு அளவு வரிசைகளில் இருப்பதால் *plot*-ஆனது மிக மிகக் குறுகிய அளவு வட்டங்களை நெருக்க நெருக்கமாக ஏற்படுத்தும். எனவே மையத்தினை சென்றடைய மிகவும் சிரமப்படும். அதுவே *normalize* செய்யப்பட்டு அனைத்து வட்டங்களும் ஒரே அளவில் இருந்தால் மையத்தினை சென்றடைய வசதியாக இருக்கும்..

Pandas

Pandas என்பது நிகழ்காலத் தரவுகளை அணுகி, அலசி நமக்கேற்றவாறு வடிவமைப்பதற்கு *python* வழங்குகின்ற ஒரு *library* ஆகும். இதன் மூலம் *csv, txt, json* போன்ற பல்வேறு வடிவங்களில் இருக்கும் மூலத் தரவுகளை எடுத்து ஒரு *dataframe*-ஆக மாற்றி நமக்கேற்றவாறு தரவுகளை தகவமைத்துக் கொள்ள முடியும்.

இங்கு நாம் பார்க்கப் போகும் உதாரணத்தில் ஒரு வீட்டின் விற்பனை விலையை நிர்ணயிப்பதற்கு உதவும் பல்வேறு காரணிகளும், அதன்படி நிர்ணயிக்கப்பட்ட விலைகளும் *csv* கோப்பாக கொடுக்கப்பட்டுள்ளன. இதுவே *training data* எனப்படும். இதை வைத்துத்தான் நாம் ஒரு *model*-ஐ உருவாக்கப்போகிறோம்.

முதலில் *model*-ஐ உருவாக்குவதற்கு முன்னர் இந்த *training data*-ஐ நாம் புரிந்து கொள்ள வேண்டும். இதில் எத்தனை தரவுகள் உள்ளன, எத்தனை *null* மதிப்புகள் உள்ளன, எவையெல்லாம் விற்பனை விலையை பாதிக்கக்கூடிய முக்கியக் காரணிகள், தேவையில்லாத இன்ன பிற காரணிகளை எவ்வாறு நீக்குவது, *Null* மதிப்புகளை எவ்வாறு நமக்கு வேண்டிய மதிப்புகளால் மாற்றி அமைப்பது போன்றவற்றையெல்லாம் *Pandas* மூலம் நாம் செய்து பார்க்கப்போகிறோம். இதுவே *preprocessing / feature selection* எனப்படும். இதற்கான நிரல் பின்வருமாறு.

https://gist.github.com/nithyadurai87/5fd84f40ce26eac65a8060ee2d15280a

```
import pandas as pd

# data can be downloaded from the url: https://www.kaggle.com/vikrishnan/boston-house-prices
df = pd.read_csv('data.csv')
target='SalePrice'

# Understanding data
```

```
print (df.shape)
print (df.columns)
print(df.head(5))
print(df.info())
print(df.describe())
print(df.groupby('LotShape').size())

# Dropping null value columns which cross the threshold
a = df.isnull().sum()
print (a)
b =  a[a>(0.05*len(a))]
print (b)
df = df.drop(b.index, axis=1)
print (df.shape)

# Replacing null value columns (text) with most used value
a1 = df.select_dtypes(include=['object']).isnull().sum()
print (a1)
print (a1.index)
for i in a1.index:
      b1 = df[i].value_counts().index.tolist()
      print (b1)
      df[i] = df[i].fillna(b1[0])

# Replacing null value columns (int, float) with most used value
a2 = df.select_dtypes(include=['integer','float']).isnull().sum()
print (a2)
b2 = a2[a2!=0].index
print (b2)
df = df.fillna(df[b2].mode().to_dict(orient='records')[0])

# Creating new columns from existing columns
print (df.shape)
a3 = df['YrSold'] - df['YearBuilt']
b3 = df['YrSold'] - df['YearRemodAdd']
df['Years Before Sale'] = a3
df['Years Since Remod'] = b3
print (df.shape)

# Dropping unwanted columns
df = df.drop(["Id", "MoSold", "SaleCondition", "SaleType", "YearBuilt",
"YearRemodAdd"], axis=1)
print (df.shape)

# Dropping columns which has correlation with target less than threshold
x = df.select_dtypes(include=['integer','float']).corr()[target].abs()
print (x)
df=df.drop(x[x<0.4].index, axis=1)
print (df.shape)

# Checking for the necessary features after dropping some columns
l1 = ["PID","MS SubClass","MS Zoning","Street","Alley","Land Contour","Lot
Config","Neighborhood","Condition 1","Condition 2","Bldg Type","House
Style","Roof Style","Roof Matl","Exterior 1st","Exterior 2nd","Mas Vnr
Type","Foundation","Heating","Central Air","Garage Type","Misc Feature","Sale
Type","Sale Condition"]
l2 = []
for i in l1:
    if i in df.columns:
        l2.append(i)

# Getting rid of nominal columns with too many unique values
for i in l2:
    len(df[i].unique())>10
    df=df.drop(i, axis=1)
print (df.columns)

df.to_csv('training_data.csv',index=False)
```

நிரலுக்கான விளக்கம் மற்றும் வெளியீடு :

csv-ல் உள்ள தரவுகள் df எனும் dataframe-க்குள் pandas மூலம் ஏற்றப்பட்டுள்ளது. இதில் எத்தனை rows மற்றும் columns உள்ளது என்பதை பின்வருமாறு அறியலாம்.

```
print (df.shape)

(1460, 81)
```

பின்வரும் கட்டளை என்னென்ன columns உள்ளது என்பதை வெளிப்படுத்தும்.

```
print (df.columns)

Index(['Id', 'MSSubClass', 'MSZoning', 'LotFrontage', 'LotArea', 'Street',
'Alley', 'LotShape', 'LandContour', 'Utilities', 'LotConfig',
'LandSlope', 'Neighborhood', 'Condition1', 'Condition2', 'BldgType',
'HouseStyle', 'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd',
'RoofStyle', 'RoofMatl', 'Exterior1st', 'Exterior2nd', 'MasVnrType',
'MasVnrArea', 'ExterQual', 'ExterCond', 'Foundation', 'BsmtQual',
'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinSF1',
'BsmtFinType2', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', 'Heating',
'HeatingQC', 'CentralAir', 'Electrical', '1stFlrSF', '2ndFlrSF',
'LowQualFinSF', 'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath',
'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr', 'KitchenQual',
'TotRmsAbvGrd', 'Functional', 'Fireplaces', 'FireplaceQu', 'GarageType',
'GarageYrBlt', 'GarageFinish', 'GarageCars', 'GarageArea', 'GarageQual',
'GarageCond', 'PavedDrive', 'WoodDeckSF', 'OpenPorchSF',
'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'PoolQC',
'Fence', 'MiscFeature', 'MiscVal', 'MoSold', 'YrSold', 'SaleType',
'SaleCondition', 'SalePrice'],
dtype='object')
```

head(5) முதல் 5 தரவுகளை வெளிப்படுத்தும்.

```
print(df.head(5))

Id MSSubClass MSZoning ... SaleType SaleCondition SalePrice
0 1 60 RL ... WD Normal 208500
1 2 20 RL ... WD Normal 181500
2 3 60 RL ... WD Normal 223500
3 4 70 RL ... WD Abnorml 140000
4 5 60 RL ... WD Normal 250000
[5 rows x 81 columns]
```

info() நமது dataframe-ன் அமைப்பு பற்றிய விவரங்களை வெளிப்படுத்தும்.

```
print(df.info())

&lt;class 'pandas.core.frame.DataFrame'&gt;
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 81 columns):
```

```
Id 1460 non-null int64
MSSubClass 1460 non-null int64
...............
SaleCondition 1460 non-null object
SalePrice 1460 non-null int64
dtypes: float64(3), int64(35), object(43)
memory usage: 924.0+ KB
None
```

describe() ஒருசில முக்கியப் புள்ளியியல் விவரங்களைக் கணக்கெடுத்து வெளிப்படுத்தும்.

```
print(df.describe())

Id MSSubClass ... YrSold SalePrice
count 1460.000000 1460.000000 ... 1460.000000 1460.000000
mean 730.500000 56.897260 ... 2007.815753 180921.195890
std 421.610009 42.300571 ... 1.328095 79442.502883
min 1.000000 20.000000 ... 2006.000000 34900.000000
25% 365.750000 20.000000 ... 2007.000000 29975.000000
50% 730.500000 50.000000 ... 2008.000000 163000.000000
75% 1095.250000 70.000000 ... 2009.000000 214000.000000
max 1460.000000 190.000000 ... 2010.000000 755000.000000
[8 rows x 38 columns]
```

groupby() ஒரு *column*-ல் உள்ள மதிப்புகளை வகைப்படுத்தி வெளிப்படுத்தும்.

```
print(df.groupby('LotShape').size())

LotShape
IR1 484
IR2 41
IR3 10
Reg 925
dtype: int64
```

ஒவ்வொரு *column*-லும் உள்ள *null* மதிப்புகளின் எண்ணிக்கையை வெளிப்படுத்தும்.

```
print (a)

Id 0
MSSubClass 0
MSZoning 0
LotFrontage 259
LotArea 0
Street 0
Alley 1369
LotShape 0
LandContour 0
Utilities 0
...............
PoolQC 1453
Fence 1179
```

```
MiscFeature 1406
MiscVal 0
MoSold 0
YrSold 0
SaleType 0
SaleCondition 0
SalePrice 0
Length: 81, dtype: int64
```

0.05 என்பது Null-க்கான threshold ஆகும். அதாவது 100 க்கு 5 null மதிப்புகள் இருக்கலாம் என வரையறுக்கப்பட்டுள்ளது. எனவே அதை விட அதிக அளவு null மதிப்புகள் கொண்ட columns கண்டறியப்பட்டு வெளிப்படுத்தப்படுகிறது. பின்னர் இவை dataframe-லிருந்து நீக்கப்படுகின்றன.

```
print (b)

LotFrontage 259
Alley 1369
MasVnrType 8
MasVnrArea 8
BsmtQual 37
BsmtCond 37
BsmtExposure 38
BsmtFinType1 37
BsmtFinType2 38
FireplaceQu 690
GarageType 81
GarageYrBlt 81
GarageFinish 81
GarageQual 81
GarageCond 81
PoolQC 1453
Fence 1179
MiscFeature 406
dtype: int64
```

மேற்கண்ட 18 columns-ஐயும் நீக்கிய பின்னர் 81 என்பது 63-ஆகக் குறைந்துள்ளத்தைக் காணலாம்.

```
print (df.shape)

(1460, 63)
```

அடுத்ததாக Threshold-ஐ விடக் குறைவான null மதிப்புகளைப் பெற்றுள்ள text column-ஆனது வெளிப்படுத்தப்படுகிறது. include=['object'] என்பது text column-ஐக் குறிக்கும்.

```
print (a1)

MSZoning 0
Street 0
LotShape 0
...............
Electrical 1
KitchenQual 0
```

```
Functional 0
PavedDrive 0
SaleType 0
SaleCondition 0
dtype: int64

print (a1.index)

Index(['MSZoning', 'Street', 'LotShape', 'LandContour', 'Utilities',
'LotConfig', 'LandSlope', 'Neighborhood', 'Condition1', 'Condition2',
'BldgType', 'HouseStyle', 'RoofStyle', 'RoofMatl', 'Exterior1st',
'Exterior2nd', 'ExterQual', 'ExterCond', 'Foundation', 'Heating',
'HeatingQC', 'CentralAir', 'Electrical', 'KitchenQual', 'Functional',
'PavedDrive', 'SaleType', 'SaleCondition'],
dtype='object')
```

அந்த columns-ல் உள்ள ஒவ்வொரு மதிப்பும் எத்தனை முறை இடம்பெற்றுள்ளது என்பது கண்டறியப்பட்டு அவை ஒரு list-ஆக மாற்றப்படுகின்றன. list-ன் முதலாவது மதிப்பு அதிக அளவு இடம்பெற்றுள்ள வார்த்தை ஆகும். இவ்வார்த்தையினால் தான் null மதிப்புகள் நிரப்பப்படுகின்றன.

```
print (b1)

['RL', 'RM', 'FV', 'RH', 'C (all)']
['Pave', 'Grvl']
['Reg', 'IR1', 'IR2', 'IR3']
...............
['Y', 'N', 'P']
['WD', 'New', 'COD', 'ConLD', 'ConLI', 'ConLw', 'CWD', 'Oth', 'Con']
['Normal', 'Partial', 'Abnorml', 'Family', 'Alloca', 'AdjLand']
```

அடுத்ததாக Threshold-ஐ விடக் குறைவான null மதிப்புகளைப் பெற்றுள்ள numerical column-ஆனது அதிக அளவு இடம்பெற்றுள்ள மதிப்பினால் நிரப்பப்படுகிறது. include=['integer','float'] என்பது numerical columns-ஐக் குறிக்கும்.

```
print (a2)

Id 0
MSSubClass 0
LotArea 0
...............
MoSold 0
YrSold 0
SalePrice 0
dtype: int64

print (b2)

Index([], dtype='object')
```

அடுத்ததாக இரண்டு column-ல் உள்ள மதிப்புகளை ஒப்பிட்டு, அவைகளின் வித்தியாசம் கண்டறியப்பட்டு ஒரு புது column-ஆக dataframe-ல் இணைக்கப்படுகிறது. 63 columns-ஆக உள்ளது புது columns இணைந்த பின் 65 என மாறியிருப்பதைக் காணலாம்.

```
print (df.shape)
(1460, 63)

print (df.shape)
(1460, 65)
```

தேவையில்லாத ஒருசில column-ன் பெயர்கள் நேரடியாகக் கொடுக்கப்பட்டு அவை dataframe-ல் இருந்து நீக்கப்படுகின்றன. பின் 59 என மாறியிருப்பதைக் காணலாம்.

```
print (df.shape)
(1460, 59)
```

numerical columns-க்கும், target columns-க்குமான correlation கண்டறியப்பட்டு வெளிப்படுத்தப்படுகிறது. இதன் மதிப்பு 0.4 எனும் threashold-ஐ விட குறைவாக இருப்பின் அவை dataframe-லிருந்து நீக்கப்படுகின்றன.

```
print (x)

MSSubClass 0.084284
LotFrontage 0.351799
LotArea 0.263843
...............
SalePrice 1.000000
Years Before Sale 0.523350
Years Since Remod 0.509079
Name: SalePrice, dtype: float64
```

மேற்கூறிய மாற்றங்கள் அனைத்தும் நிகழ்ந்த பின், நமக்குத் தேவையான ஒருசில முக்கிய விஷயங்கள் dataframe-ல் இன்னும் உள்ளதா என்பது சோதிக்கப்படுகிறது.அளவுக்கு அதிகமான தனிப்பட்ட மதிப்புகளைக் கொண்ட columns நீக்கப்படுகின்றன. இவையும் நீக்கப்பட்டபின் columns எண்ணிக்கை 38 என மாறியிருப்பதைக் காணலாம்.

```
print (df.shape)
(1460, 38)
```

பின்னர் அவை எந்தெந்த columns என வெளிப்படுத்தப்படுகின்றன.

```
print (df.columns)

Index(['MSZoning', 'LotShape', 'LandContour', 'Utilities', 'LotConfig',
'LandSlope', 'Condition1', 'Condition2', 'BldgType', 'HouseStyle',
'OverallQual', 'RoofStyle', 'RoofMatl', 'Exterior1st', 'Exterior2nd',
'ExterQual', 'ExterCond', 'TotalBsmtSF', 'HeatingQC', 'CentralAir',
'Electrical', '1stFlrSF', 'GrLivArea', 'FullBath', 'KitchenQual',
```

```
'TotRmsAbvGrd', 'Functional', 'Fireplaces', 'GarageCars', 'GarageArea',
'PavedDrive', 'SalePrice', 'Years Before Sale', 'Years Since Remod'],
dtype='object')
```

கடைசியாக இந்த dataframe-ல் இருக்கும் மதிப்புகளானது training_data எனும் பெயரில் .csv கோப்பாக சேமிக்கப்படுகின்றன. இதுவே model-ன் உருவாக்கத்திற்கு உள்ளீடாக அமையும். இதை வைத்து model-ஐ உருவாக்குவது எப்படி என்று அடுத்த பகுதியில் காணலாம்.

Model file handling

Model Creation

sklearn (sk for scikit) என்பது python-ல் உள்ள இயந்திரவழிக் கற்றலுக்கான ஒரு library ஆகும். இதில் classification, regression ஆகிய வகைகளின் கீழ் அமையும் linear, ensemble, neural networks போன்ற அனைத்து விதமான model-க்கும் algorithms காணப்படும். இதிலிருந்து LinearRegression எனும் algorithm-ஐ எடுத்து அதற்கு நம்முடைய data-வைப் பற்றி நாம் கற்றுத் தருகிறோம். இதற்கான நிரல் பின்வருமாறு.

https://gist.github.com/nithyadurai87/91e74160ccb4ff51eef3188372a78b91

```
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split,cross_val_score
from sklearn.externals import joblib
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
from math import sqrt
import os

df = pd.read_csv('./training_data.csv')

i = list(df.columns.values)
i.pop(i.index('SalePrice'))
df0 = df[i+['SalePrice']]
df = df0.select_dtypes(include=['integer','float'])
print (df.columns)

X = df[list(df.columns)[:-1]]
y = df['SalePrice']
X_train, X_test, y_train, y_test = train_test_split(X, y)
regressor = LinearRegression()
regressor.fit(X_train, y_train)

y_predictions = regressor.predict(X_test)

meanSquaredError=mean_squared_error(y_test, y_predictions)
rootMeanSquaredError = sqrt(meanSquaredError)

print("Number of predictions:",len(y_predictions))
print("Mean Squared Error:", meanSquaredError)
print("Root Mean Squared Error:", rootMeanSquaredError)
print ("Scoring:",regressor.score(X_test, y_test))
```

```
plt.plot(y_predictions,y_test,'r.')
plt.plot(y_predictions,y_predictions,'k-')
plt.title('Parity Plot - Linear Regression')
plt.show()

plot = plt.scatter(y_predictions, (y_predictions - y_test), c='b')
plt.hlines(y=0, xmin= 100000, xmax=400000)
plt.title('Residual Plot - Linear Regression')
plt.show()

joblib.dump(regressor, './salepricemodel.pkl')
```

நிரலுக்கான வெளியீடு:

```
Index(['OverallQual', 'TotalBsmtSF', '1stFlrSF', 'GrLivArea', 'FullBath',
'TotRmsAbvGrd', 'Fireplaces', 'GarageCars', 'GarageArea',
'Years Before Sale', 'Years Since Remod', 'SalePrice'],
dtype='object')
Number of predictions: 365
Mean Squared Error: 981297922.7884247
Root Mean Squared Error: 31325.675136993053
Scoring: 0.818899237738355
```

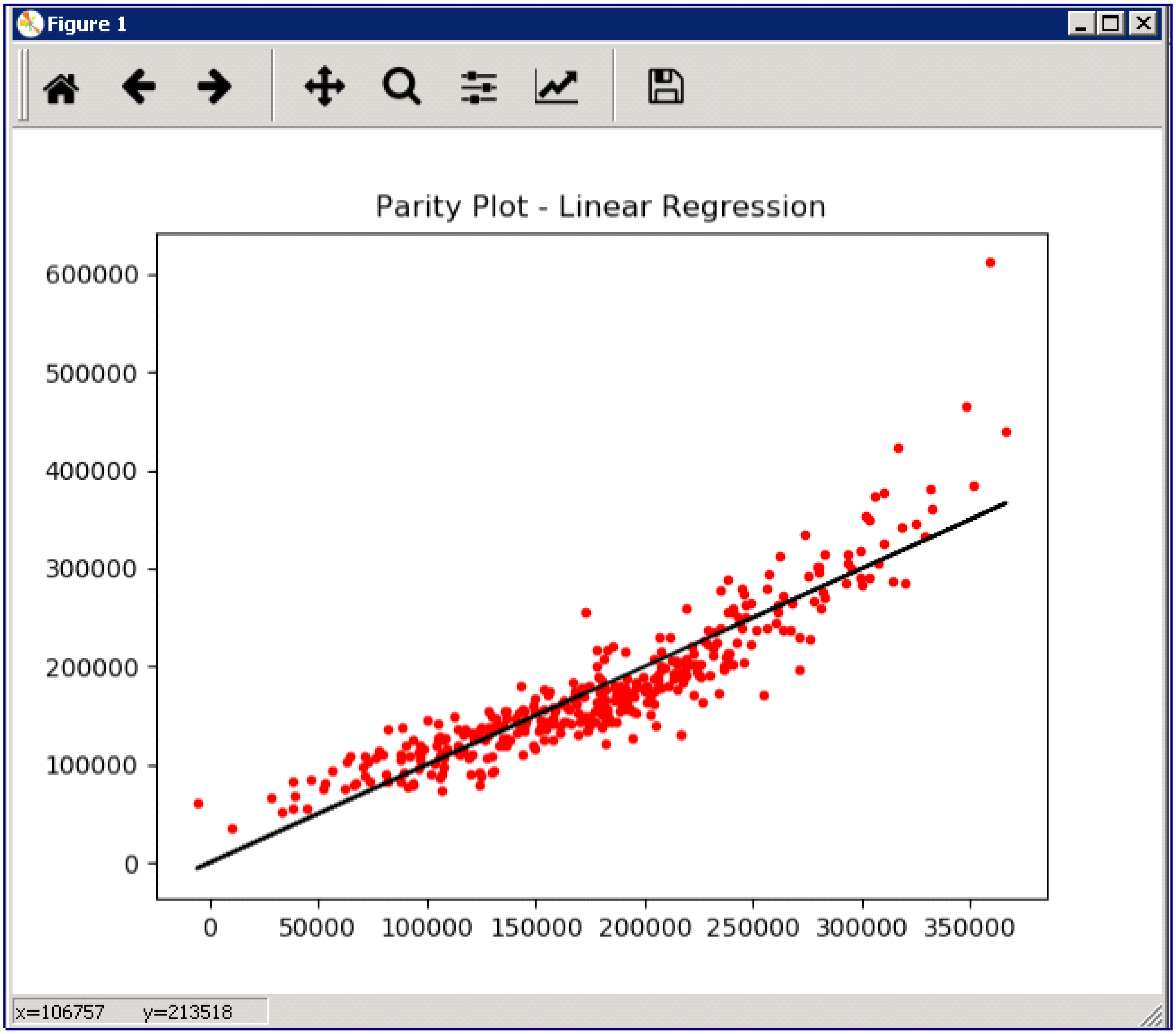
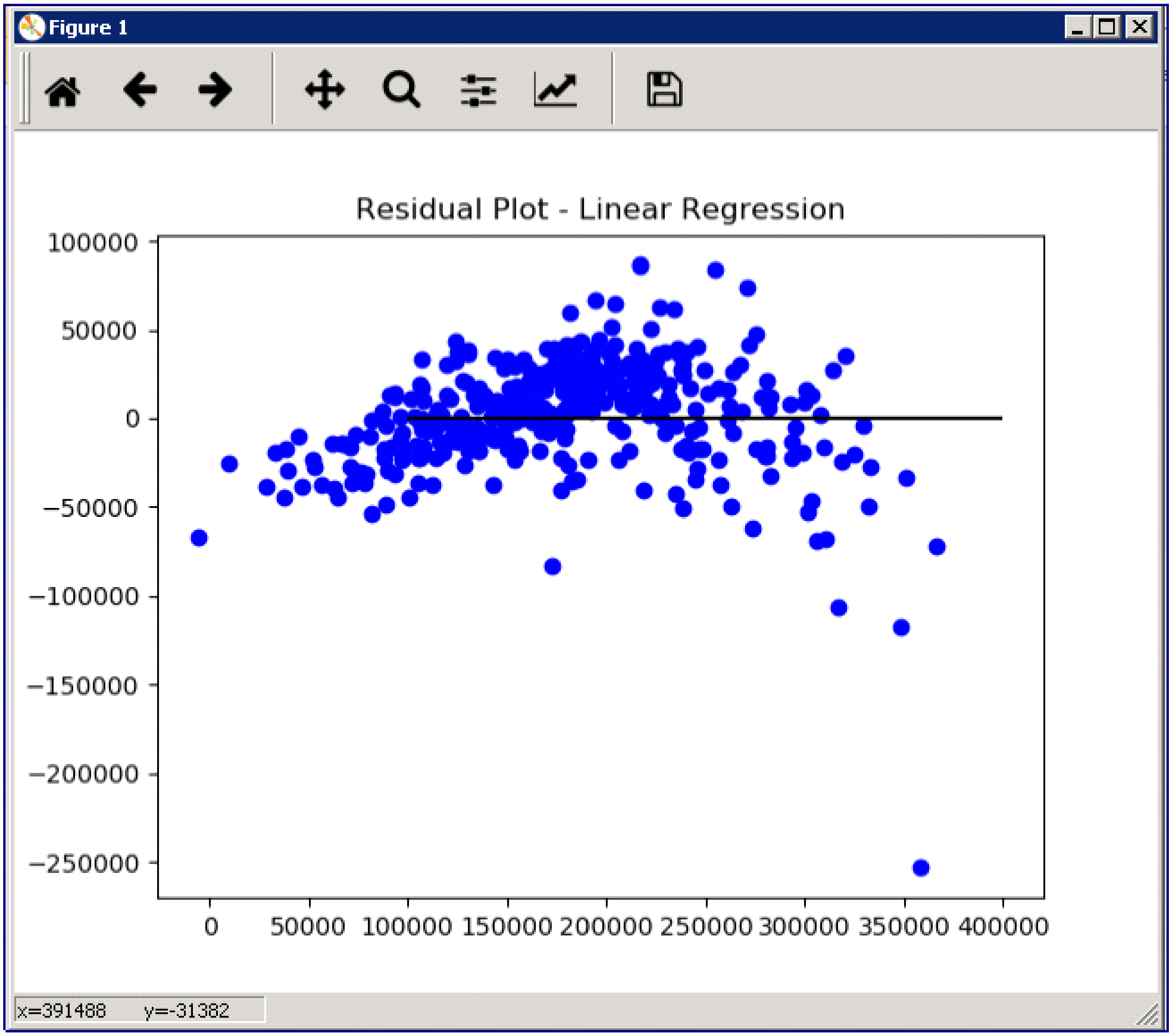


Figure 1
Parity Plot - Linear Regression
600000
500000
400000
300000
200000
100000
0
0
50000
100000
150000
200000
250000
300000
350000
x=106757 y=213518



<u>நிரலுக்கான விளக்கம்:</u>

1. *training_data எனும் கோப்பிற்குள் உள்ள அனைத்துத் தரவுகளும் df-க்குள் செலுத்தப்பட்டுவிட்டன.*
2. *கணிப்பதற்கு உதவும் அனைத்தும் X-லும், கணிக்கப்பட வேண்டிய 'SalePrice' என்பது y-லும் சேமிக்கப்பட்டுள்ளது. இதற்கு முன்னர் pop() என்பது கணிக்கப்பட வேண்டிய column-ஐ df-லிருந்து நீக்கி பின்னர் மீண்டும் கடைசி column-ஆக இணைக்கிறது. இதன் மூலம் [:-1] எனக் கொடுத்து கடைசிக்கு முன்னால் உள்ள அனைத்தும் X-லும் கடைசி column-ஆன 'SalePrice'-ஐ y-லும் சேமித்துக் கொள்ளலாம்.*
3. *fit() என்பது கற்றுக் கொடுப்பதற்கும், predict() என்பது கணிப்பதற்கும் பயன்படுகிறது.*
4. *score() என்பது நமது algorithm எவ்வளவு தூரம் சரியாகக் கற்றுக்கொண்டுள்ளது என்பதை மதிப்பிடப் பயன்படுகிறது.*
5. *train_test_split() என்பது நம்முடைய மொத்தத் தரவுகளை 75% - 25% எனும் விகிதத்தில்*

பிரிக்கிறது. அதாவது 75% தரவுகள் கற்றுக் கொடுப்பதற்கும், 25% தரவுகள் சோதனை செய்து மதிப்பிடுவதற்கும் பயன்படும்.

6. mean_squared_error, sqrt ஆகிய functions, நமது algorithm-ஆல் கணிக்கப்படும் மதிப்புகளுக்கும் உண்மையான மதிப்புகளுக்கும் உள்ள இழப்பின் சராசரியைக் கண்டறிந்து கூறும். இந்த இழப்பு தான் 'Residual Error' ஆகும். இது ஒரு வரைபடமாக வரைந்து காட்டப்பட்டுள்ளது.
7. joblib என்பது நமது model-ஐ .pkl கோப்பாக சேமிக்கும். இதுவே pickle file ஆகும். இது serialization மற்றும் de-serialization-க்கு உதவுகின்ற ஒரு binary கோப்பு வகை ஆகும். இதை வைத்து எவ்வாறு புதிய தரவுகளை கணிப்பது என அடுத்த பகுதியில் காணலாம்.

Prediction

நமது கோப்பில் உள்ள முதல் தரவினை மட்டும் கொடுத்து அதற்கான விலையை கணிக்கச் சொல்லுவோம். இது input.json எனும் கோப்பின் வழியே கொடுக்கப்படுகிறது..

```
cat input.json

{
"OverallQual":[7],
"TotalBsmtSF":[856],
"1stFlrSF":[856],
"GrLivArea":[1710],
"FullBath":[2],
"TotRmsAbvGrd":[8],
"Fireplaces":[0],
"GarageCars":[2],
"GarageArea":[548],
"Years Before Sale":[5],
"Years Since Remod":[5]
}
```

predict() செய்வதற்கான நிரல் பின்வருமாறு.

https://gist.github.com/nithyadurai87/4a31b465220448ab05b84d2227e4e8a5

```
import os
import json
import pandas as pd
import numpy
from sklearn.externals import joblib

s = pd.read_json('./input.json')
p = joblib.load("./salepricemodel.pkl")
r = p.predict(s)

print (str(r))
```

நிரலுக்கான வெளியீடு:

```
[213357.65598157]
```

உண்மையான SalePrice மதிப்பு 208500 எனில் நமது நிரல் 213357 எனும் மதிப்பினை வெளிப்படுத்தும். இது கிட்டத்தட்ட பரவாயில்லை. ஏனெனில் நமது algorithm-ன் score, 81% ஆகும். எனவே இந்த அளவு வித்தியாசம் இருக்கத்தான் செய்யும்.

நிரலுக்கான விளக்கம்:

1. *joblib.load() என்பது binary வடிவில் உள்ள கோப்பினை de-serialize செய்து algorithm-ஆக மாற்றி சேமிக்கும்.*
2. *பின்னர் இதன் மீது செயல்படும் predict() ஆனது json வடிவில் உள்ள தரவுகளை உள்ளீடாகக் கொடுத்து அதற்கான வெளியீட்டினைக் கணிக்கிறது..*

அடுத்த இந்த prediction-க்கான உள்ளீடு மற்றும் வெளியீட்டு மதிப்பினை எவ்வாறு ஒரு Rest API-ஆக expose செய்வது என்று பார்க்கலாம்.

Flask API

நமது algorithm கணிக்கும் மதிப்பினை ஒரு API-ஆக expose செய்வதற்கு Flask பயன்படுகிறது. இதற்கான நிரல் பின்வருமாறு.

https://gist.github.com/nithyadurai87/9d04097e006e2fe6c7a96b1da643cb3a

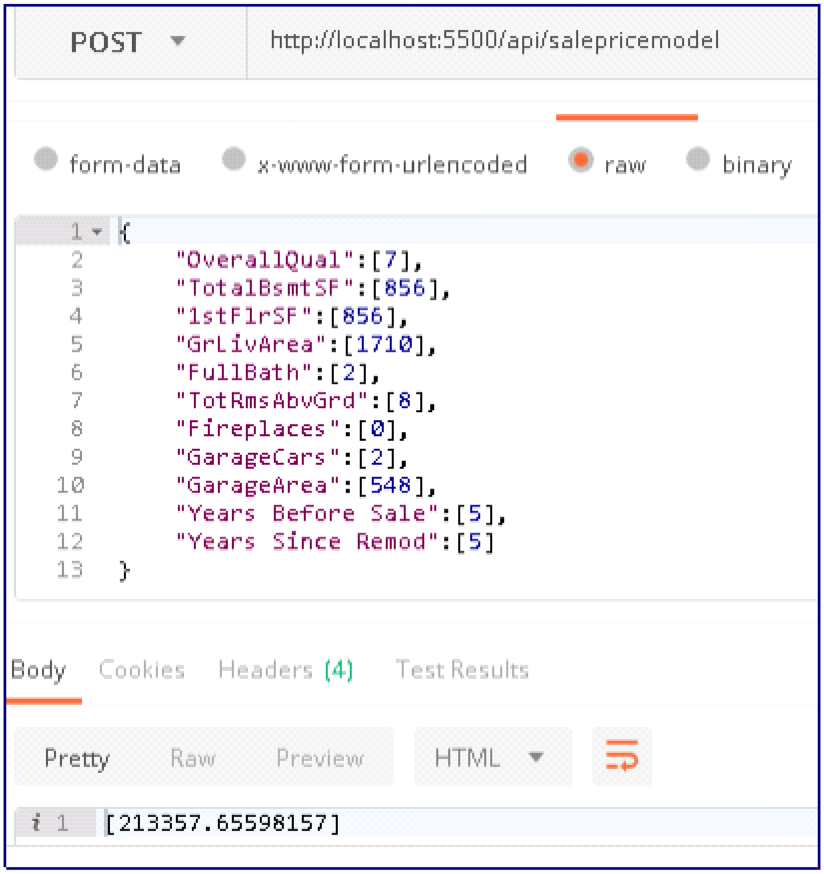
```
import os
import json
import pandas as pd
import numpy
from flask import Flask, render_template, request, jsonify
from pandas.io.json import json_normalize
from sklearn.externals import joblib

app = Flask(__name__)
port = int(os.getenv('PORT', 5500))

@app.route('/')
def home():
    return render_template('index.html')

@app.route('/api/salepricemodel', methods=['POST'])
def salepricemodel():
    if request.method == 'POST':
        try:
            post_data = request.get_json()
            json_data = json.dumps(post_data)
            s = pd.read_json(json_data)
            p = joblib.load("./salepricemodel.pkl")
            r = p.predict(s)
            return str(r)

        except Exception as e:
            return (e)
```

```
if __name__ == '__main__':
app.run(host='0.0.0.0', port=port, debug=True)
```

நிரலுக்கான வெளியீடு:

** Serving Flask app "flask_api" (lazy loading)*
** Environment: production*
WARNING: Do not use the development server in a production environment.
Use a production WSGI server instead.
** Debug mode: on*
** Restarting with stat*
** Debugger is active!*
** Debugger PIN: 690-746-333*
** Running on http://0.0.0.0:5500/ (Press CTRL+C to quit)*

இதனை *postman* எனும் கருவி மூலம் நாம் சோதித்துக் கொள்ளலாம்.



Model comparison

நமது model உருவாக்கத்திற்கு வெறும் linear regression-ஐ மட்டும் பயன்படுத்தாமல், வேறு சில algorithm-வுடனும் ஒப்பிட்டு எது சிறந்ததோ அதை பயன்படுத்த வேண்டும். இதற்கான நிரல் பின்வருமாறு. இது நமது தரவுகளை பல்வேறு algorithm-ல் பொருத்தி, ஒவ்வொன்றினுடைய Score மற்றும் RMSE மதிப்புகளை வெளிப்படுத்துகிறது. இவற்றில் சிறந்ததை நாம் தேர்வு செய்து கொள்ளலாம்.

https://gist.github.com/nithyadurai87/9ecfcbf04593d245e26316d52b0708e1

```
import pandas as pd
from sklearn.linear_model import LinearRegression, Ridge, Lasso, ElasticNet
from sklearn.ensemble import RandomForestRegressor, AdaBoostRegressor,
ExtraTreesRegressor, GradientBoostingRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.neural_network import MLPRegressor
from sklearn.model_selection import train_test_split,cross_val_score
from sklearn.externals import joblib
from sklearn.metrics import mean_squared_error
from azure.storage.blob import BlockBlobService
import matplotlib.pyplot as plt
from math import sqrt
import numpy as np
import os

df = pd.read_csv('./training_data.csv')

i = list(df.columns.values)
i.pop(i.index('SalePrice'))
df0 = df[i+['SalePrice']]
df = df0.select_dtypes(include=['integer','float'])

X = df[list(df.columns)[:-1]]
y = df['SalePrice']
X_train, X_test, y_train, y_test = train_test_split(X, y)

def linear():
      regressor = LinearRegression()
      regressor.fit(X_train, y_train)
      y_predictions = regressor.predict(X_test)
      return (regressor.score(X_test, y_test),sqrt(mean_squared_error(y_test,
y_predictions)))

def ridge():
      regressor = Ridge(alpha=.3, normalize=True)
      regressor.fit(X_train, y_train)
      y_predictions = regressor.predict(X_test)
      return (regressor.score(X_test, y_test),sqrt(mean_squared_error(y_test,
y_predictions)))

def lasso():
      regressor = Lasso(alpha=0.00009, normalize=True)
      regressor.fit(X_train, y_train)
      y_predictions = regressor.predict(X_test)
      return (regressor.score(X_test, y_test),sqrt(mean_squared_error(y_test,
y_predictions)))

def elasticnet():
      regressor = ElasticNet(alpha=1, l1_ratio=0.5, normalize=False)
      regressor.fit(X_train, y_train)
```

```
      y_predictions = regressor.predict(X_test)
      return (regressor.score(X_test, y_test),sqrt(mean_squared_error(y_test, 
y_predictions)))

def randomforest():
      regressor = 
RandomForestRegressor(n_estimators=15,min_samples_split=15,criterion='mse',max_de
pth=None)
      regressor.fit(X_train, y_train)
      y_predictions = regressor.predict(X_test)
      print("Selected Features for RamdomForest",regressor.feature_importances_)
      return (regressor.score(X_test, y_test),sqrt(mean_squared_error(y_test, 
y_predictions)))

def perceptron():
      regressor = MLPRegressor(hidden_layer_sizes=(5000,), activation='relu', 
solver='adam', max_iter=1000)
      regressor.fit(X_train, y_train)
      y_predictions = regressor.predict(X_test)
      print("Co-efficients of Perceptron",regressor.coefs_)
      return (regressor.score(X_test, y_test),sqrt(mean_squared_error(y_test, 
y_predictions)))

def decisiontree():
      regressor = DecisionTreeRegressor(min_samples_split=30,max_depth=None)
      regressor.fit(X_train, y_train)
      y_predictions = regressor.predict(X_test)
      print("Selected Features for DecisionTrees",regressor.feature_importances_)
      return (regressor.score(X_test, y_test),sqrt(mean_squared_error(y_test, 
y_predictions)))

def adaboost():
      regressor = AdaBoostRegressor(random_state=8, 
loss='exponential').fit(X_train, y_train)
      regressor.fit(X_train, y_train)
      y_predictions = regressor.predict(X_test)
      print("Selected Features for Adaboost",regressor.feature_importances_)
      return (regressor.score(X_test, y_test),sqrt(mean_squared_error(y_test, 
y_predictions)))

def extratrees():
      regressor = ExtraTreesRegressor(n_estimators=50).fit(X_train, y_train)
      regressor.fit(X_train, y_train)
      y_predictions = regressor.predict(X_test)
      print("Selected Features for Extratrees",regressor.feature_importances_)
      return (regressor.score(X_test, y_test),sqrt(mean_squared_error(y_test, 
y_predictions)))

def gradientboosting():
      regressor = GradientBoostingRegressor(loss='ls',n_estimators=500, 
min_samples_split=15).fit(X_train, y_train)
      regressor.fit(X_train, y_train)
      y_predictions = regressor.predict(X_test)
      print("Selected Features for 
Gradientboosting",regressor.feature_importances_)
      return (regressor.score(X_test, y_test),sqrt(mean_squared_error(y_test, 
y_predictions)))

print ("Score, RMSE values")
print ("Linear = ",linear())
print ("Ridge = ",ridge())
print ("Lasso = ",lasso())
print ("ElasticNet = ",elasticnet())
print ("RandomForest = ",randomforest())
print ("Perceptron = ",perceptron())
print ("DecisionTree = ",decisiontree())
print ("AdaBoost = ",adaboost())
print ("ExtraTrees = ",extratrees())
print ("GradientBoosting = ",gradientboosting())
```

நிரலுக்கான வெளியீடு.:

Score, RMSE values
Linear = (0.7437086925668539, 40067.32048747698)
Ridge = (0.7426559924644496, 40149.523137601194)
Lasso = (0.7437086997392647, 40067.31992682729)
ElasticNet = (0.7427716507607811, 40140.499909601196)
RandomForest = (0.7816174352942802, 36985.57224959144)
Perceptron = (0.7090884723574984, 42687.80529374248)
DecisionTree = (0.7205230305007451, 41840.45264436496)
AdaBoost = (0.7405881117926998, 40310.51057481991)
ExtraTrees = (0.8112271823246542, 34386.90514804029)
GradientBoosting = (0.770865727419495, 37885.095662535474)

Selected Features for RamdomForest [0.61070268 0.04279095 0.04336447 0.17066371 0.01107406 0.01329107 0.0065515 0.03938371 0.02458596 0.02051551 0.01707638]

Selected Features for DecisionTrees [0.75618387 0.03596786 0.02304119 0.13037245 0.0022674 0. 0.00739768 0.01056845 0.01184136 0.01171254 0.01064719]

Selected Features for Adaboost [0.38413232 0.18988447 0.03844386 0.12826885 0.03857277 0.03995005 0.01059839 0.08066205 0.05036717 0.01473333 0.02438674]

Selected Features for Extratrees [0.33168574 0.04675749 0.05913052 0.11159271 0.05178125 0.02947481 0.03966461 0.16786223 0.06241882 0.05316226 0.04646956]

Selected Features for Gradientboosting [0.04426232 0.16359645 0.14768597 0.25403034 0.02119119 0.04361512 0.01825781 0.01626673 0.15891844 0.07188963 0.06028599]

Co-efficients of Perceptron [array([[2.83519650e-01, 7.33024272e-03, 2.80373628e-01, ..., -1.43939606e-03, -3.84913926e-02],
[1.34495184e-01, 1.31687141e-02, 1.72078666e-04, ...,1.70666499e-23, -2.31494718e-02, -1.08758545e-02],
[9.44490485e-02, -2.34835375e-02, 2.37798999e-02, ..., -1.74549692e-02, -2.70192753e-02, -3.67706290e-02],
...,
[1.59527225e-01, -3.19744701e-02, -1.22884400e-01, ..., -2.35994429e-26, -3.03880584e-02, -2.85251050e-02],
[-3.63149939e-01, -4.05674884e-02, 2.66679331e-01, ..., -1.73628910e-02, 7.40224353e-03, -6.89871249e-03],
[-4.30743882e-01, 7.07948777e-03, 3.34518179e-01, ..., -1.74075111e-02, 3.47755293e-02, -2.64627071e-02]]),

array([[0.16789784],[-0.01864141],[0.20432696],...,[0.01739125],[-0.02779454],[-0.00476935]])]

ஒரு model-ஐ இதுவே சிறந்தது எனக் கூறுவதற்கு, அதனுடைய Score மற்றும் RMSE மதிப்பு அல்லாது Threshold Limit, Sensitivity போன்றவற்றையும் நாம் கணக்கில் கொள்ள வேண்டும். இதைப் பற்றியும் மேலே குறிப்பிட்டுள்ள ஒவ்வொரு algorithm-ஐப் பற்றியும் பின்னர் நாம் விளக்கமாகக் காணலாம். மேலே குறிப்பிட்டுள்ள algorithms-ல் ஒருசிலவை எந்தெந்த features-ஐ வைத்து கணித்துள்ளது என்பதை வெளிப்படுத்தியுள்ளது. ஆனால் இந்தப் பண்பு linear, ridge, lasso, elasticnet போன்றவற்றிற்குக் கிடையாது. ஆகவே இதுபோன்ற algorithms-க்கு RFE technique மூலம் நாம் features-ஐ தேர்வு செய்து அனுப்ப வேண்டும். இதைப் பற்றி 'feature selection' எனும் அடுத்த பகுதியில் காணலாம்.

Improving Model score

நாம் உருவாக்கிய model-ன் score-ஆனது மிகவும் குறைவாக இருக்கிறது எனில், அது எந்த இடத்தில் அதிகம் வேறுபடுகிறது எனக் கண்டறிய trend / parity போன்ற வரைபடங்களைப் போட்டுப் பார்க்க வேண்டும். கீழ்க்கண்ட உதாரணத்தில் ஒரு வீட்டின் விலையை நிர்ணயிப்பதற்கான பல்வேறு அம்சங்களும், அதனடிப்படையில் நிர்ணயிக்கப்பட்ட விற்பனை விலைகளும் பயிற்சிக்குக் கொடுக்கப்பட்டுள்ளன. இதை வைத்து நாம் உருவாக்கிய model-ன் score ஆனது 35 என வந்துள்ளது. எனவே எந்த இடத்தில் உண்மையான விலையும், கணிக்கப்படும் விலையும் அதிகம் வேறுபடுகிறது எனக் கண்டறிய trend, parity plots வரையப்பட்டுள்ளன.

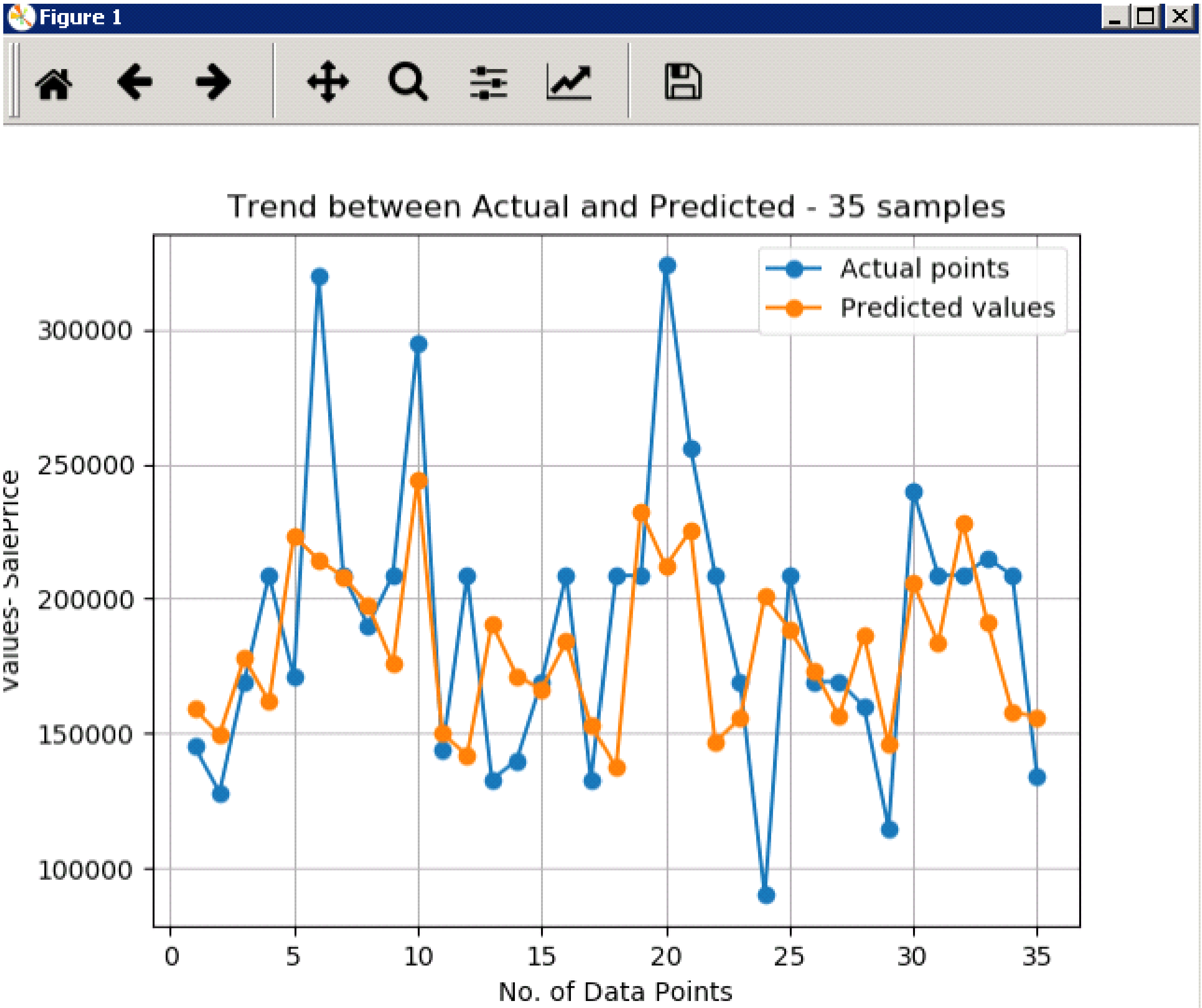
நிரல் மற்றும் அதன் வெளியீடு:

https://gist.github.com/nithyadurai87/ca54a4a8f59187cb988b5145d000c70c

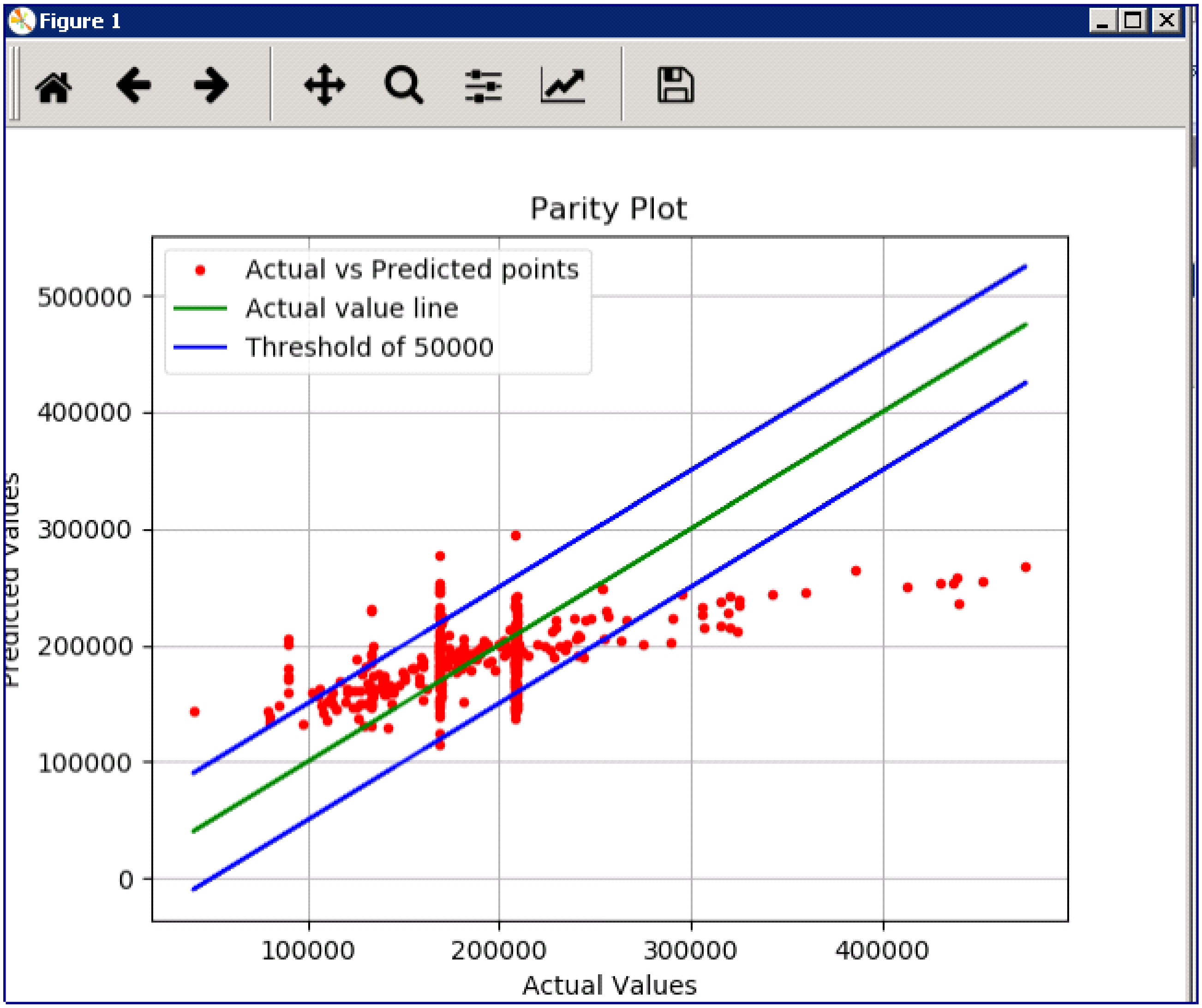
```
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split,cross_val_score
from sklearn.externals import joblib
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
from math import sqrt
import os

df = pd.read_csv('./training_data.csv')

X = df[list(df.columns)[:-1]]
y = df['SalePrice']
X_train, X_test, y_train, y_test = train_test_split(X, y)
regressor = LinearRegression()
regressor.fit(X_train, y_train)

y_predictions = regressor.predict(X_test)

meanSquaredError=mean_squared_error(y_test, y_predictions)
rootMeanSquaredError = sqrt(meanSquaredError)

print("Number of predictions:",len(y_predictions))
print("Mean Squared Error:", meanSquaredError)
print("Root Mean Squared Error:", rootMeanSquaredError)
print ("Scoring:",regressor.score(X_test, y_test))

## TREND PLOT
y_test25 = y_test[:35]
y_predictions25 = y_predictions[:35]
myrange = [i for i in range(1,36)]
fig = plt.figure()
ax = fig.add_subplot(111)
ax.grid()
plt.plot(myrange,y_test25, marker='o')
plt.plot(myrange,y_predictions25, marker='o')
plt.title('Trend between Actual and Predicted - 35 samples')
ax.set_xlabel("No. of Data Points")
ax.set_ylabel("Values- SalePrice")
plt.legend(['Actual points','Predicted values'])
plt.savefig('TrendActualvsPredicted.png',dpi=100)
plt.show()


## PARITY PLOT
y_testp = y_test[:]+50000
y_testm = y_test[:]-50000
```

```
fig = plt.figure()
ax = fig.add_subplot(111)
ax.grid()
plt.plot(y_test,y_predictions,'r.')
plt.plot(y_test,y_test,'k-',color = 'green')
plt.plot(y_test,y_testp,color = 'blue')
plt.plot(y_test,y_testm,color = 'blue')
plt.title('Parity Plot')
ax.set_xlabel("Actual Values")
ax.set_ylabel("Predicted Values")
plt.legend(['Actual vs Predicted points','Actual value line','Threshold of 
50000'])
plt.show()

## Data Distribution
fig = plt.figure()
plt.plot([i for i in range(1,1461)],y,'r.')
plt.title('Data Distribution')
plt.show()

a, b = 0 , 0
for i in range(0,1460):
    if(y[i]>250000):
        a += 1
    else:
        b +=1
print(a, b)

#X = X[:600]
#y = y[:600]
```

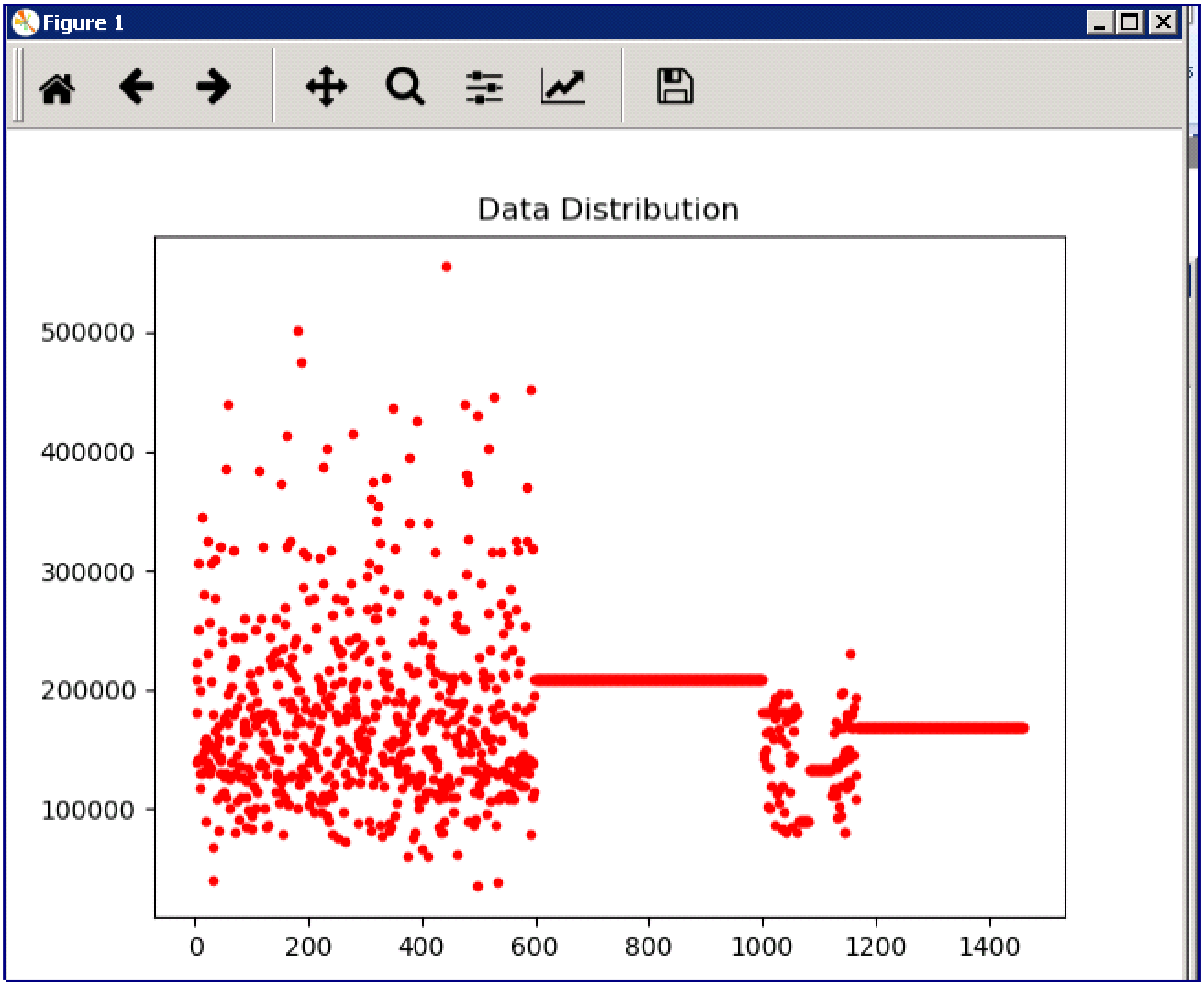
Trend plot என்பது உண்மையான விலைகளும் model-கணித்த விலைகளும் எந்த அளவுக்கு வித்தியாசப்படுகின்றன என்பதைக் காட்டுகிறது.



Parity plot என்பது அந்த வித்தியாசத்திற்கு ஒரு threshold-ஐ அமைக்கிறது. அதாவது விலை வேறுபாடானது 50 ஆயிரம் வரை முன்னும் பின்னும் செல்லலாம் எனக் கொடுத்து அந்த threshold-க்குள் எவ்வளவு விலைகள் அமைந்துள்ளன, அதற்கு மேல் எவ்வளவு அமைந்துள்ளது என்பதைக் காட்டுகிறது.



அடுத்தபடியாக data distribution chart வரையப்பட்டுள்ளது. இதன் X-அச்சில் பயிற்சிக்கு அளிக்கப்பட்டுள்ள 1460 rows-ம், Y-அச்சில் விற்பனை விலைகளும் வரைபடமாக வரைந்து காட்டப்பட்டுள்ளன. இதில் முதல் 600 records-வரை விற்பனை விலைகள் 1 லட்சத்திலிருந்து 5 லட்சம் வரை பரவலாகப் பரவியுள்ளத்தைக் காணலாம். அதற்கு மேல் 600-லிருந்து 1000 records-வரை விற்பனை விலைகள் அனைத்தும் வெறும் 2 லட்சத்திலேயே அதிகம் அமைந்திருப்பதைக் காணலாம். இதுவே model-ன் குறைந்த அளவு score-க்குக் காரணம். பயிற்சி அளிக்கப்படும் தரவுகளானது சீரான முறையில் பரவலாக அமைந்திருக்க வேண்டும் என ஏற்கனவே கண்டோம். இங்கு அவ்வாறு இல்லை. எனவே எதுவரை சீராகப் பரவியுள்ளதோ அதுவரை மட்டும் உள்ள தரவுகளைக் கொடுத்து model-ஐ உருவாக்கும்போது அதன் score அதிகரிப்பதைக் காணலாம். X = df[list(df.columns)[:-1]] , y = df['SalePrice'] எனக் கொடுத்த பின்னர், X = X[:600] , y = y[:600] எனும் வரிகளை இணைத்தால் போதுமானது. முதல் 600 records வரை மட்டும் உள்ள தரவுகளை எடுத்து நமது model உருவாக்கப்படும்.



<u>Output:</u>

Number of predictions: 365

Mean Squared Error: 2312162517.277571

Root Mean Squared Error: 48084.95104788578

Scoring: 0.34729555622354125

97 1363

கடைசியாக பயிற்சிக்குக் கொடுக்கப்பட்டுள்ள 1460 தரவுகளில் 250000-க்கும் மேல் எவ்வளவு மதிப்புகள் உள்ளன, அதற்குக் கீழ் எவ்வளவு மதிப்புகள் உள்ளன என்பது கண்டுபிடிக்கப்பட்டுள்ளது. இதில் 1363 மதிப்புகள் 250000-க்கு கீழும், வெறும் 97 மதிப்புகள் அதற்கு மேலும் அமைந்துள்ளன. எனவே இதுவும் சீராக இல்லை. இதுவே outliers எனப்படுகிறது. இதுபோன்ற outliers-ஐ எவ்வாறு கண்டுபிடித்து நீக்குவது என அடுத்த பகுதியில் காணலாம்.

Feature Selection

ஒரு கோப்பினுள் பல்வேறு *columns* இருக்கிறதெனில், அவற்றுள் எந்தெந்த *column* மதிப்புகளைப் பொறுத்து நாம் கணிக்கின்ற விஷயம் அமைகிறது எனக் கண்டுபிடிப்பதே *feature selection* ஆகும். உதாரணத்துக்கு *400, 500 columns*-ஐக் கொண்டுள்ள கோப்பிலிருந்து, *prediction*-க்கு உதவும் ஒருசில முக்கிய *columns*-ஐத் தேர்வு செய்வது *feature selection* ஆகும். இதற்கு முதலில் நம்மிடமுள்ள *columns*-ஐ *process variables, manipulated variables & disturbance variables* எனும் *3* வகையின் கீழ் பிரிக்க வேண்டும். இதில் *manipulated* மற்றும் *disturbance* இரண்டும் *input*-க்கான *parameter*-ஆகவும், *process* என்பது *output*-க்கான *parameter*-ஆகவும் அமைகிறது.

- *Manipulated Variables (MV)* - இவ்வகையின் கீழ் அமையும் *columns*-ல் உள்ள மதிப்புகளை நம்மால் மாற்றி அமைக்க முடியும். நமக்கேற்றவாறு இதனை நாம் கையாளலாம்.
- *Disturbance Variables (DV)* - இதனை நம்மால் நேரடியாக மாற்றி அமைக்க முடியாது. ஆனால் *manipulated*-ன் மதிப்பினைப் பொறுத்தே இதன் மதிப்பு அமைகிறது.
- *Process Variales (PV)* - பல்வேறு செயல்முறைகளைப் பொறுத்து இதிலுள்ள மதிப்புகள் அமையும். அதாவது மற்ற *columns* மதிப்புகளைப் பொறுத்தே இதன் மதிப்பு அமைகிறது. எனவே இதில் நாம் மாற்றுவதற்கு எதும் கிடையாது .

மேற்கண்டவாறு பிரித்த பின் ஒவ்வொரு *variable*-க்கும் மற்ற *variables*-வுடன் இருக்கும் தொடர்பினைக் கணக்கிடல் வேண்டும். இதுவே *correlation* எனப்படும். இதன் மதிப்பு *-1* லிருந்து *+1* வரை அமையும். *-1* என்பது எதிர்மறைத் தொடர்பையும், *+1* நேர்மறைத் தொடர்பையும் குறிக்கும்.

உதாரணத்துக்கு "உண்ணும் உணவின் அளவு", "உடற்பயிற்சி செய்யும் நேரம்", "எடை குறைப்பு புத்தகங்களைப் படிக்கும் நேரம்" போன்ற சில பல *features*-ஐ வைத்து, "உடலின் எடை" எனும் ஒரு விஷயத்தை நாம் கணிக்கப் போவதாகக் கொண்டால் அதற்கான *correlation matrix* ல் உள்ள மதிப்புகள் பின்வருமாறு அமையும்.

- *Positive Correlation:* உடலின் எடைக்கும் - உண்ணும் உணவின் அளவுக்குமான தொடர்பு *+1* என வெளிப்படும். உணவின் அளவு அதிகரித்தால் எடை அதிகரிக்கும்.
- *Negative Correlation:* உடலின் எடைக்கும் - உடற்பயிற்சி செய்யும் நேரத்திற்குமான தொடர்பு *-1* என வெளிப்படும். உடற்பயிற்சி செய்யும் நேரம் அதிகரித்தால் உடலின் எடை குறையும்.
- *Zero Correlation:* எடை குறைப்பு பற்றிய புத்தகங்களைப் படிக்கும் நேரத்துடன் கொண்டுள்ள தொடர்பு *0* என வெளிப்படும். படிக்கும் நேரத்திற்க்கும் உடலின் எடைக்கும் யாதொரு சம்மந்தமும் இல்லை
- இவையல்லாத வேறு சில *features* இருப்பின் அவை கொண்டுள்ள தொடர்பினைப் பொறுத்து, அதற்கான மதிப்பு *-1* லிருந்து *1* வரை அமையும்.

Highly Correlated features (MV - DV)

கீழ்க்கண்ட உதாரணத்தில், *data.csv* எனும் கோப்பிற்குள் உள்ள *columns*-ல் எது எது என்னென்ன வகையான *parameters* எனும் விவரத்தை நாம் *domain expert*-ன் உதவி கொண்டு தெரிந்து

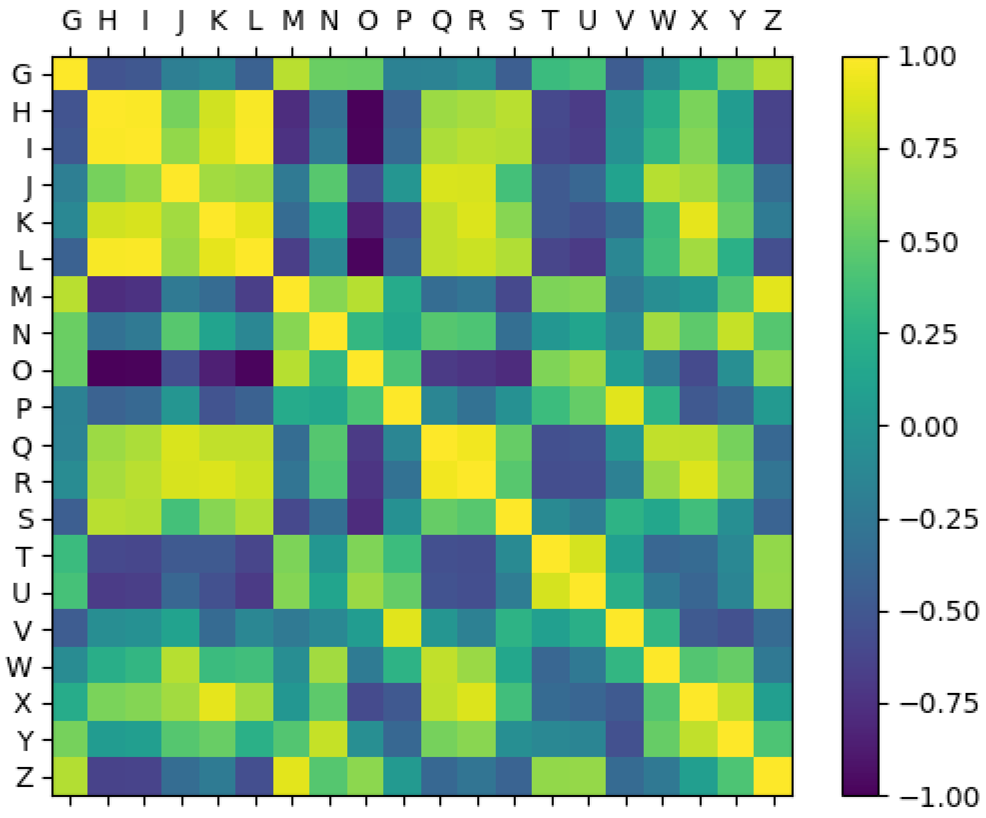
கொள்ளலாம். உதாரணத்துக்கு *A* முதல் *Z* வரை பெயர்கள் கொண்ட *26 features*-ல் *A,B,C,D,E,F* ஆகியவை *process parameters* ஆகவும், மற்றவை *manipulated* மற்றும் *disturbance parameters* ஆகவும் கருதியுள்ளோம். எனவே முதலில் *process parameters* அனைத்தும் *dataframe*-லிருந்து நீக்கப்படுகின்றன. பின்னர் மீதியுள்ள *manipulated* மற்றும் *disturbance parameters*-க்கான *correlation* கண்டுபிடிக்கப்பட்டு, அது கோப்பு வடிவிலும், வரைபட வடிவிலும் வெளிப்படுத்தப்பட்டுள்ளது. இவற்றில் அதிக அளவு நேர்மறை மற்றும் எதிர்மறைத் தொடர்பு கொண்டுள்ளவை *dataframe*-லிருந்து நீக்கப்படுகின்றன. அதாவது *-98,-99,-1,98,99,1* எனும் தொடர்பினைப் பெற்றிருக்கும் இரு *features*-ல் ஒன்று நீக்கப்படுகிறது. இவ்வாறாக *manipulated* மற்றும் *disturbance*-க்கிடையில் அதிகத் தொடர்பு கொண்டுள்ள அம்சங்கள் கண்டுபிடிக்கப்பட்டு அவற்றில் ஒன்று நீக்கப்படுகிறது. மீதமுள்ள அனைத்தும் *training_data* எனும் பெயரில் சேமிக்கப்படுகிறது. இதுவே நமது *process variable*-க்கும், தேர்ந்தெடுக்கப்பட்ட *manipulated & disturbance variable*-க்குமான தொடர்பினைக் கண்டறிவதற்கு உள்ளீடாக அமைகிறது. இவைகள் அனைத்தும் நாம் கணிக்க வேண்டிய *process variable*-வுடன் கொண்டுள்ள தொடர்பினைக் கண்டுபிடித்து, அதில் *0* தொடர்பு பெற்றுள்ள *columns*-ஐ நீக்குவது அடுத்த படியாக அமைகிறது.

https://gist.github.com/nithyadurai87/5a43155d33cf5288204def23661704d0

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split,cross_val_score
from sklearn.metrics import mean_squared_error
from math import sqrt
from sklearn.feature_selection import RFE
from sklearn.datasets import make_friedman1

df = pd.read_csv('./data.csv')

# Dropping all process parameters
df = df.drop(["A","B", "C", "D", "E", "F"], axis=1)

#finding correlation between manipulated & disturbance variables
correlations = df.corr()
correlations = correlations.round(2)
correlations.to_csv('MV_DV_correlation.csv',index=False)
fig = plt.figure()
g = fig.add_subplot(111)
cax = g.matshow(correlations, vmin=-1, vmax=1)
fig.colorbar(cax)
ticks = numpy.arange(0,20,1)
g.set_xticks(ticks)
g.set_yticks(ticks)
g.set_xticklabels(list(df.columns))
g.set_yticklabels(list(df.columns))
plt.savefig('MV_DV_correlation.png')

#removing parameters with high correlation
upper = correlations.where(numpy.triu(numpy.ones(correlations.shape),
k=1).astype(numpy.bool))
cols_to_drop = []
for i in upper.columns:
      if (any(upper[i] == -1) or any(upper[i] == -0.98) or any(upper[i] == -0.99)
or any(upper[i] == 0.98) or any(upper[i] == 0.99) or any(upper[i] == 1)):
            cols_to_drop.append(i)
df = df.drop(cols_to_drop, axis=1)

print (df.shape,df.columns)
df.to_csv('./training_data.csv',index=False)
```

நிரலுக்கான வெளியீடு:

(20, 17) Index(['G', 'H', 'J', 'K', 'M', 'N', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z'], dtype='object')



G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
1	-0.52	-0.5	-0.19	-0.11	-0.42	0.78	0.53	0.52	-0.16	-0.15	-0.08	-0.44	0.33	0.39	-0.46	-0.08	0.2	0.57	0.76
-0.52	1	0.99	0.57	0.85	0.98	-0.77	-0.3	-1	-0.41	0.69	0.72	0.78	-0.6	-0.68	-0.07	0.22	0.58	0.06	-0.64
-0.5	0.99	1	0.66	0.87	0.99	-0.74	-0.23	-0.99	-0.36	0.74	0.78	0.76	-0.61	-0.67	-0.03	0.29	0.61	0.09	-0.63
-0.19	0.57	0.66	1	0.71	0.68	-0.23	0.46	-0.57	0.01	0.88	0.87	0.38	-0.48	-0.38	0.12	0.77	0.71	0.45	-0.33
-0.11	0.85	0.87	0.71	1	0.92	-0.34	0.13	-0.85	-0.52	0.8	0.89	0.62	-0.48	-0.54	-0.35	0.33	0.92	0.52	-0.22
-0.42	0.98	0.99	0.68	0.92	1	-0.67	-0.12	-0.98	-0.42	0.8	0.83	0.75	-0.62	-0.69	-0.12	0.36	0.71	0.24	-0.56
0.78	-0.77	-0.74	-0.23	-0.34	-0.67	1	0.62	0.77	0.18	-0.33	-0.27	-0.6	0.59	0.61	-0.23	-0.07	0.02	0.44	0.91
0.53	-0.3	-0.23	0.46	0.13	-0.12	0.62	1	0.3	0.15	0.45	0.42	-0.32	0.02	0.14	-0.11	0.71	0.48	0.82	0.45
0.52	-1	-0.99	-0.57	-0.85	-0.98	0.77	0.3	1	0.41	-0.69	-0.72	-0.78	0.6	0.68	0.07	-0.22	-0.58	-0.06	0.64
-0.16	-0.41	-0.36	0.01	-0.52	-0.42	0.18	0.15	0.41	1	-0.13	-0.29	-0.03	0.34	0.5	0.9	0.26	-0.49	-0.37	0.04
-0.15	0.69	0.74	0.88	0.8	0.8	-0.33	0.45	-0.69	-0.13	1	0.96	0.51	-0.55	-0.53	0.01	0.8	0.79	0.57	-0.37
-0.08	0.72	0.78	0.87	0.89	0.83	-0.27	0.42	-0.72	-0.29	0.96	1	0.46	-0.57	-0.56	-0.17	0.68	0.89	0.63	-0.27
-0.44	0.78	0.76	0.38	0.62	0.75	-0.6	-0.32	-0.78	-0.03	0.51	0.46	1	-0.1	-0.21	0.26	0.15	0.36	-0.05	-0.4
0.33	-0.6	-0.61	-0.48	-0.48	-0.62	0.59	0.02	0.6	0.34	-0.55	-0.57	-0.1	1	0.86	0.1	-0.38	-0.35	-0.11	0.66
0.39	-0.68	-0.67	-0.38	-0.54	-0.69	0.61	0.14	0.68	0.5	-0.53	-0.56	-0.21	0.86	1	0.23	-0.24	-0.39	-0.14	0.67
-0.46	-0.07	-0.03	0.12	-0.35	-0.12	-0.23	-0.11	0.07	0.9	0.01	-0.17	0.26	0.1	0.23	1	0.29	-0.48	-0.54	-0.35
-0.08	0.22	0.29	0.77	0.33	0.36	-0.07	0.71	-0.22	0.26	0.8	0.68	0.15	-0.38	-0.24	0.29	1	0.44	0.51	-0.24
0.2	0.58	0.61	0.71	0.92	0.71	0.02	0.48	-0.58	-0.49	0.79	0.89	0.36	-0.35	-0.39	-0.48	0.44	1	0.8	0.09
0.57	0.06	0.09	0.45	0.52	0.24	0.44	0.82	-0.06	-0.37	0.57	0.63	-0.05	-0.11	-0.14	-0.54	0.51	0.8	1	0.42
0.76	-0.64	-0.63	-0.33	-0.22	-0.56	0.91	0.45	0.64	0.04	-0.37	-0.27	-0.4	0.66	0.67	-0.35	-0.24	0.09	0.42	1

Zero Correlated features (PV - MV,DV)

"A" என்பது நாம் கணிக்க வேண்டிய process parameter எனக் கொள்வோம். training_data எனும் கோப்பிற்குள், இந்த "A" -வை கடைசி column-ஆக இணைத்து கீழ்க்கண்ட நிரலுக்கு உள்ளீடாக அனுப்பவும். பின்னர் A-க்கும் மற்ற parameters-க்குமான தொடர்பினைக் கண்டுபிடித்து, அதில் 0 தொடர்பு கொண்டுள்ள MV, DV -யை நிக்கிவிடவும். இங்கு 0.6 -க்கும் குறைவான அதாவது 0.1, 0.2, 0.3, 0.4, 0.5 எனும் மதிப்புகளைப் பெற்றுள்ள columns நீக்கப்படுகின்றன.

https://gist.github.com/nithyadurai87/e0cca6ec864405a032888244122a90d8

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split,cross_val_score
from sklearn.metrics import mean_squared_error
from math import sqrt
from sklearn.feature_selection import RFE
from sklearn.datasets import make_friedman1

df = pd.read_csv('./training_data.csv')
print (df.shape,df.columns)

# Dropping columns which has correlation with target less than threshold
target = "A"
correlations = df.corr()[target].abs()
correlations = correlations.round(2)
correlations.to_csv('./PV_MVDV_correlation.csv',index=False)
df=df.drop(correlations[correlations<0.06].index, axis=1)

print (df.shape,df.columns)
df.to_csv('./training.csv',index=False)
```

நிரலுக்கான வெளியீடு:

(20, 18) Index(['G', 'H', 'J', 'K', 'M', 'N', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z', 'A'], dtype='object')

(20, 17) Index(['G', 'H', 'J', 'K', 'M', 'N', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'A'], dtype='object')

	A
1	0.14
2	0.32
3	0.84
4	0.62
5	0.06
6	0.78
7	0.14
8	0.87
9	0.85
10	0.14
11	0.4
12	0.33
13	0.16
14	0.87
15	0.79
16	0.81
17	0.05
18	1

Recursive Feature Elimination Technique

இது *RFE technique* என்று அழைக்கப்படும். *Randomforest, Decisiontree, Adaboost, Extratrees, gradient boosting* போன்ற *algorithms* தானாகவே *features*-ஐ தேர்வு செய்யும் அளவுக்கு திறன் பெற்று விளங்கும். ஆனால், *linear regression, ridge, lasso, elasticnet* போன்ற *algorithms*-க்கு இத்தகைய *techniques* மூலம் நாம் தான் *features*-ஐ தேர்வு செய்து வழங்க வேண்டும். இந்த நுட்பமானது ஒரு *algorithm*-ஐ உள்ளீடாகப் பெற்றுக் கொண்டு, ஒவ்வொரு *feature*-க்கும் *ranking*-ஐ வழங்குகிறது.. இதில் *rank 1* பெற்றுள்ள *feature*-ஐ மட்டும் தேர்வு செய்து நாம் பயன்படுத்தலாம்.

https://gist.github.com/nithyadurai87/34ca5b0e8a9f5908276240eb099247ad

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import train_test_split,cross_val_score
from sklearn.metrics import mean_squared_error
from math import sqrt
from sklearn.feature_selection import RFE
from sklearn.datasets import make_friedman1
```

```
df = pd.read_csv('./training.csv')

X = df[list(df.columns)[:-1]]
y = df['A']
X_train, X_test, y_train, y_test = train_test_split(X, y)

regressor = DecisionTreeRegressor(min_samples_split=3,max_depth=None)
regressor.fit(X_train, y_train)
y_predictions = regressor.predict(X_test)
print ("Selected Features for DecisionTree",regressor.feature_importances_)

# RFE Technique - Recursive Feature Elimination
X, y = make_friedman1(n_samples=20, n_features=17, random_state=0)
selector = RFE(LinearRegression())
selector = selector.fit(X, y)
print ("Selected Features for LinearRegression",selector.ranking_)
```

இங்கு *feature_importances_* எனும் *method, decisiontree*-ன் மீது செயல்பட்டு, அனைத்து *features*-க்குமான *ranking*-ஐ வெளிப்படுத்தியுள்ளதைக் காணலாம். ஆனால் இந்த *method, linear regression* மீது செயல்படாது. எனவே *RFE* மூலம் நாம்தான் *ranking*-ஐ வெளிப்படுத்துமாறு செய்ய வேண்டும். பின்னர் இதிலிருந்து *Rank 1* வெளிப்பட்டுள்ள *features*-ஐ மட்டும் தேர்வு செய்து பயன்படுத்தலாம்.

<u>நிரலுக்கான வெளியீடு:</u>

Selected Features for DecisionTree [9.52359304e-04 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 6.15147906e-03 2.23327627e-03 7.70622020e-02
0.00000000e+00 0.00000000e+00 1.10263284e-03 2.33946020e-04
0.00000000e+00 0.00000000e+00 9.12264104e-01 0.00000000e+00]
Selected Features for LinearRegression [1 1 10 1 1 9 8 3 5 2 6 7 1
1 1 4 1]

Outliers Removal

Outlier என்பது மற்ற தரவுகளிலிருந்து வேறுபட்டு சற்று தள்ளி இருக்கும் தரவு ஆகும். *5,10,15,20...75* எனும் மதிப்பினைக் கொண்டிருக்கும் தரவு வரிசைகளில் ஒன்றே ஒன்று மட்டும் *15676* எனும் எண்ணைக் கொண்டிருப்பின், அதுவே *outlier* ஆகும். இதைத் தான் நாம் கண்டறிந்து களைய வேண்டும்.

கீழ்க்கண்ட உதாரணத்தில், உள்ளீடாக உள்ள கோப்பிற்குள் இருக்கும் *outliers* ஒவ்வொரு *column*-லும் கண்டறியப்பட்டு அவை ஒரு வரைபடமாக வெளிப்படுத்தப்படுகின்றன. *boxplot* அல்லது *violinplot* இதற்குப் பயன்படுகின்றன.

https://gist.github.com/nithyadurai87/1756b2a5ec421fc3f36add04909cc517

```
import pandas as pd
import pylab
import numpy as np
from scipy import stats
from scipy.stats import kurtosis
from scipy.stats import skew
```

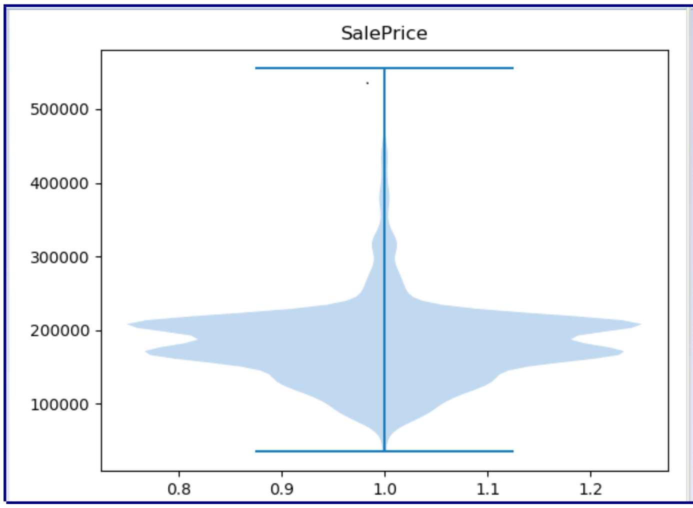
```
import matplotlib._pylab_helpers

df = pd.read_csv('./14_input_data.csv')

# Finding outlier in data
for i in range(len(df.columns)):
    pylab.figure()
    pylab.boxplot(df[df.columns[i]])
    #pylab.violinplot(df[df.columns[i]])
    pylab.title(df[df.columns[i]].name)

list1=[]

for i in matplotlib._pylab_helpers.Gcf.get_all_fig_managers():
    list1.append(i.canvas.figure)
print (list1)

for i, j in enumerate(list1):
    j.savefig(df[df.columns[i]].name)

# Removing outliers
z = np.abs(stats.zscore(df))
print(z)
print(np.where(z > 3))
print(z[53][9])
df1 = df[(z < 3).all(axis=1)]
print (df.shape)
print (df1.shape)
```

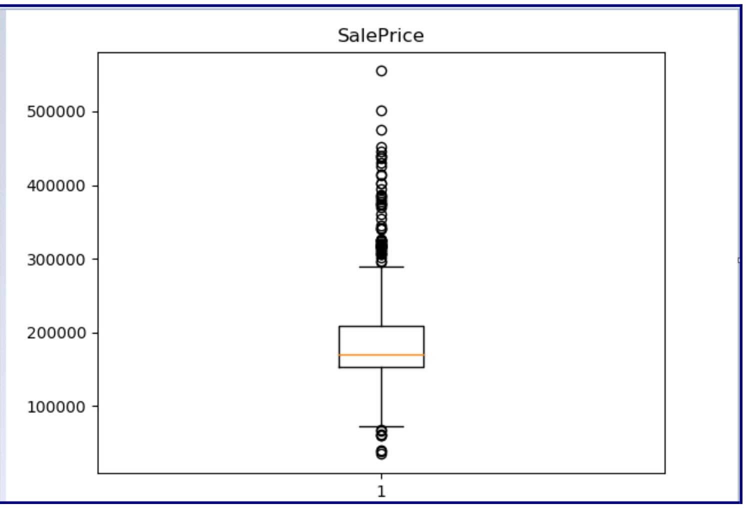
list1 என்பதற்குள் ஒவ்வொரு column-க்குமான வரைபடங்கள் சேமிக்கப்பட்டுகின்றன.

print(list1)

*[<Figure size 640x480 with 1 Axes>, <Figure size 640x480 with 1 Axes>, <Figure s
ize 640x480 with 1 Axes>, <Figure size 640x480 with 1 Axes>, <Figure size 640x480 with 1 Axes>, <Figure size 640x48
0 with 1 Axes>, <Figure size 640x480 with 1 Axes>, <Figure size 640x480 with 1 A
xes>, <Figure size 640x480 with 1 Axes>, <Figure size 640x480 with 1 Axes>, <Fig
ure size 640x480 with 1 Axes>]*

பின்னர் savefig() மூலம் ஒவ்வொரு column-க்குமான வரைபடமும் அதன் பெயரிலேயே சேமிக்கப்படுகிறது. கீழ்க்கண்ட படங்களில் இடது பக்கம் இருப்பது 'salePrice' -க்கான violin plot ஆகும். வலது பக்கம் இருப்பது அதே column-க்கான box plot ஆகும். இவற்றில் ஏதாவது ஒன்றைப் பயன்படுத்தி outlier இருக்கும் இடத்தை நாம் தெரிந்து கொள்ளலாம். இங்கு SalePrice-ல் 300000-க்கு மேலும் 100000-க்கு கீழும் outlier இருப்பதாக வெளிப்படுத்தியுள்ளது.





அடுத்தபடியாக இத்தகைய outliers-ஐ எவ்வாறு நீக்குவது என்று பார்க்கலாம். Z Score, IQR Score போன்றவை இதற்காகப் பயன்படுகின்றன. Z Score என்பது ஒரு தரவு அதற்கான mean மதிப்பிலிருந்து எவ்வளவு தூரம் தள்ளி இருக்கிறது என்பதைக் கணக்கிட்டுக் கூறும். அதிக அளவு தள்ளி இருப்பவற்றை நாம் outlier-ஆக எடுத்துக் கொள்ளலாம்.

0 என்பதனை mean-ஆக வைத்துக்கொண்டு, அதிலிருந்து ஒவ்வொரு தரவும் எந்த அளவுக்கு தள்ளி உள்ளது என்பது பின்வருமாறு.

```
print(z)
```

```
[[0.65147924 0.45930254 0.79343379 ... 0.31172464 0.35100032 0.4732471 ]
[0.07183611 0.46646492 0.25714043 ... 0.31172464 0.06073101 0.01235858]
[0.65147924 0.31336875 0.62782603 ... 0.31172464 0.63172623 0.74302803]
...
[0.65147924 0.21564122 0.06565646 ... 1.02685765 1.03391416 0.23194227]
[0.79515147 0.04690528 0.21898188 ... 1.02685765 1.09005935 0.23192429]
[0.79515147 0.45278362 0.2416147 ... 1.02685765 0.9216238 0.2319063 ]]
```

வெறும் மேற்கண்ட மதிப்பினை மட்டும் வைத்துக்கொண்டு, outliers-ஐ சொல்லி விட முடியாது. அதற்கு ஒரு threshold-ஐ அமைக்க வேண்டும். பொதுவாக 3 என்பது threshold-ஆக அமையும். அதாவது 3-க்கும் மேல் தள்ளி இருப்பவை எல்லாம் outliers ஆகும். எனவே இந்த outliers-ஐ மட்டும் print செய்வதற்கான கட்டளை பின்வருமாறு.

```
print(np.where(z > 3))
```

```
(array([ 53, 58, 112, 118, 151, 161, 166, 178, 178, 185, 185,
185, 197, 224, 224, 224, 231, 278, 304, 309, 309, 313,
321, 332, 336, 349, 375, 378, 389, 440, 440, 440, 473,
477, 481, 496, 496, 496, 496, 515, 523, 523, 523, 527,
529, 533, 581, 585, 591, 605, 608, 635, 635, 642, 664,
691, 691, 691, 769, 769, 798, 803, 825, 897, 898, 910,
1024, 1031, 1044, 1044, 1061, 1169, 1173, 1182, 1182, 1182, 1190,
1230, 1268, 1298, 1298, 1298, 1298, 1298, 1298, 1350, 1353, 1373,
1373, 1386], dtype=int64), array([9, 9, 9, 3, 9, 9, 6, 8, 9, 3, 5, 9, 3,
1, 2, 9, 9, 9, 3, 6, 9, 9,
9, 1, 9, 9, 0, 9, 9, 1, 2, 9, 9, 9, 9, 1, 2, 3, 9, 9, 1, 2, 3, 9,
2, 0, 8, 9, 9, 6, 3, 3, 5, 6, 8, 1, 2, 3, 3, 5, 3, 5, 8, 5, 2, 5,
2, 5, 1, 2, 8, 3, 5, 1, 2, 3, 8, 5, 3, 1, 2, 3, 5, 6, 8, 5, 3, 1,
```

2, 5], dtype=int64))

மேற்கண்ட வெளியீட்டில் இரண்டு arrays() உள்ளத்தைக் கவனிக்கவும். இதன் முதல் array()-ல் outlier அமைந்துள்ள இடத்தின் row மதிப்பும், இரண்டாவது array()-ல் அதன் column-மதிப்பும் காணப்படும். எனவே print(z[53][9]) எனக் கொடுக்கும்போது 53-வது row, 9-வது column-ல் உள்ள z core மதிப்பு 3.647669390284779 என வெளிப்படுவதைக் காணலாம்.

கடைசியாக 3-க்குக் கீழ் உள்ள மதிப்புகள் மட்டும் ஒரு புதிய dataframe-ல் சேமிக்கப்பட்டு அவையே outliers நீக்கப்பட்ட தரவுகளாக சேமிக்கப்படுகின்றன.

df1 = df[(z < 3).all(axis=1)]

எனவே பழைய dataframe-ல் 1460 rows இருப்பதையும், புதிய dataframe-ல் 1396 rows இருப்பதையும் காணலாம்.

(1460, 10)
(1396, 10)

Explanatory Data Analysis

நமது தரவுகள் எவ்வாறு அமைந்துள்ளன என விரிவாக ஆராய்ந்து பார்ப்பதே Explanatory Data Analysis ஆகும்.

Univariate

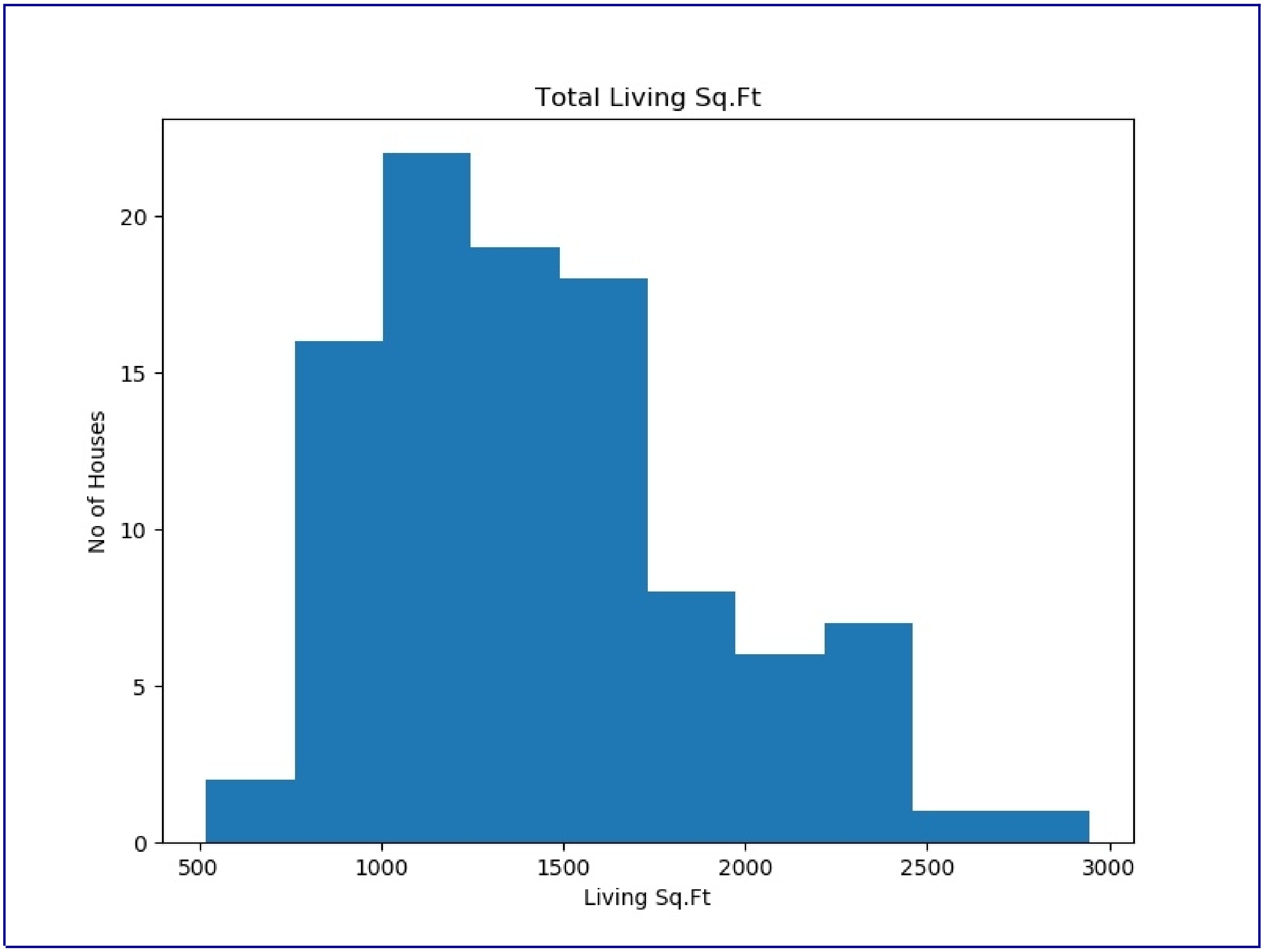
ஒரே ஒரு column-ல் உள்ள தரவுகளை மட்டும் எடுத்து ஆராய்வது univariate எனவும், இரண்டு column-ல் உள்ளவை எவ்விதத்தில் ஒன்றோடொன்று தொடர்பினை ஏற்படுத்துகின்றன என ஆராய்வது bivariate எனவும், பல்வேறு columns இணைந்து எவ்வாறு ஒரு target column-ன் மீது தாக்கத்தை ஏற்படுத்துகிறது எனப் பார்ப்பது multi-variate analysis எனவும் அழைக்கப்படும்.

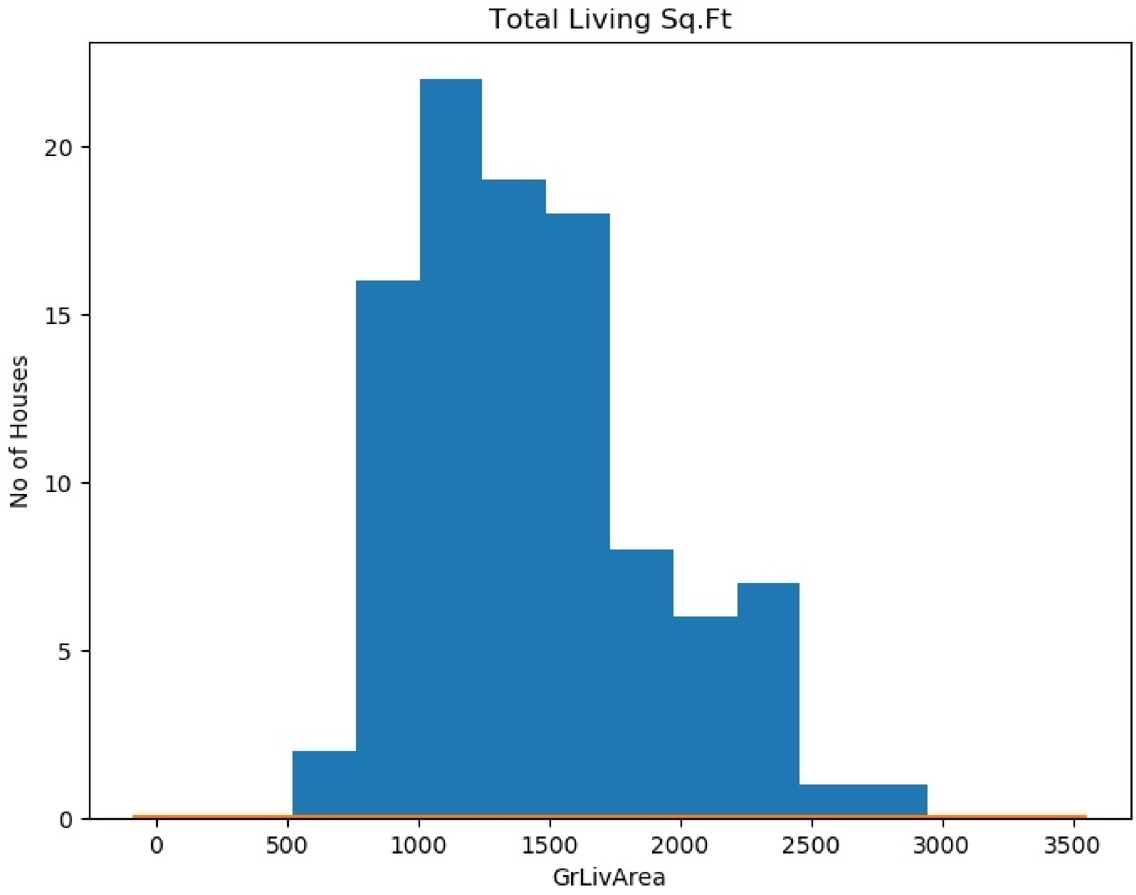
histogram, Density plot மற்றும் box plot ஆகியவை univariate analysis-க்கு பெரிதும் உதவுகின்ற வரைபட வகைகள் ஆகும். Histogram என்பது ஒரு variable-ல் உள்ளவற்றை, பல்வேறு bins-ஆகப் பிரித்து, ஒவ்வொரு bin-லும் எவ்வாறு தரவுகள் அமைந்துள்ளன என்பதைக் காட்டுகிறது. கீழ்க்கண்ட உதாரணத்தில், 'GrLivArea' எனும் column-ல் பல்வேறு வீட்டினுடைய sqft அளவுகள் கொடுக்கப்பட்டுள்ளன. அவை 500, 1000, 1500 ... 3000 எனும் பல்வேறு bins-ஆகப் பிரிக்கப்பட்டு, ஒவ்வொரு bin-லும் எத்தனை வீடுகள் அமைந்துள்ளன என்பது வரைபடமாகக் காட்டப்பட்டுள்ளது. matplotlib மற்றும் seaborn ஆகியவை இத்தகைய வரைபடங்களை வழங்குகின்றன. Histogram என்பது matplotlib வழங்குகின்ற வரைபடமெனில், Densityplot என்பது seaborn வழங்குகின்ற வரைபடம் ஆகும்.

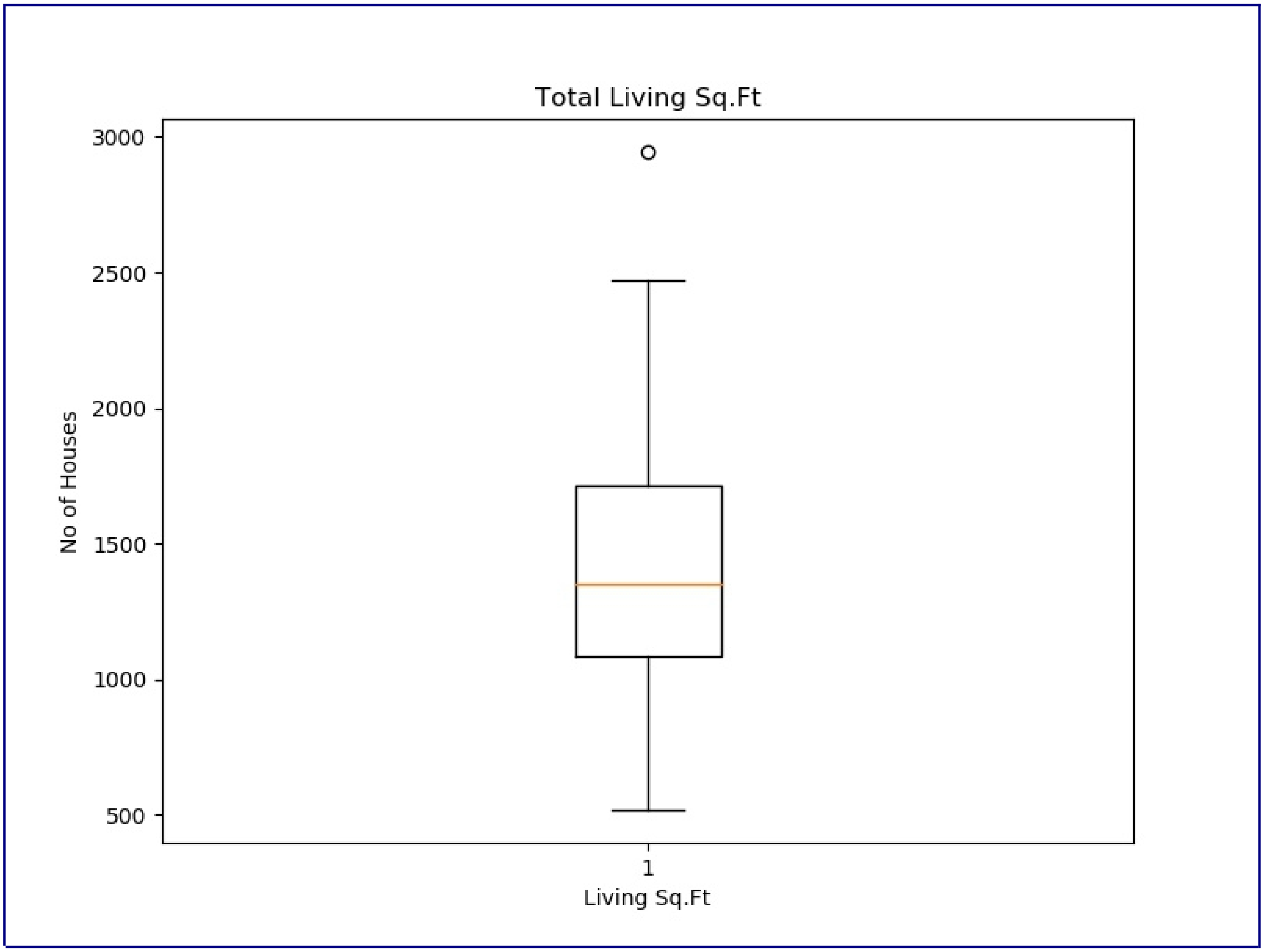
Boxplot என்பதும் ஒரே ஒரு variable-ஐ analysis செய்வதற்கு உதவும் ஒரு வரைபட வகை ஆகும். இதில் ஒரு பெட்டி போன்ற படம் ஒன்று காணப்படும். இதன் நடுவில் உள்ள கோடு தான் median ஆகும். இந்தப் பெட்டிக்கு மேலும், கீழும் உள்ள கோடு, எந்த அளவுக்கு தரவுகள் பரவியுள்ளது என்பதைக் காட்டும். அந்தக் கோட்டின் எல்லையையும் தாண்டி ஆங்காங்கு காணப்படும் ஒரு சில சிறிய புள்ளிகளே outliers ஆகும்.

https://gist.github.com/nithyadurai87/5be067164741348c6a51d6af6d8d78b7

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

df = pd.read_csv("14_input_data.csv")
df = df.fillna(0)
df = df[:100]

y = [i for i in range(0,10)]
fig = plt.figure(figsize=(8,6))
ax = fig.add_subplot(111)
ax.set(title="Total Living Sq.Ft",
      ylabel='No of Houses', xlabel='Living Sq.Ft')
ax.hist(df['GrLivArea'])
plt.savefig('Histogram.jpg')

sns.distplot(df['GrLivArea'], hist = False, kde = True,
                 kde_kws = {'shade': True, 'linewidth': 3})
plt.savefig('DensityPlot.jpg')

fig = plt.figure(figsize=(8,6))
ax = fig.add_subplot(111)
ax.set(title="Total Living Sq.Ft",
      ylabel='No of Houses', xlabel='Living Sq.Ft')
ax.boxplot(df['GrLivArea'])
plt.savefig('BoxPlot.jpg')
```


Total Living Sq.Ft
No of Houses
0
5
10
15
20
500
1000
1500
2000
2500
3000
Living Sq.Ft


Total Living Sq.Ft
No of Houses
0
5
10
15
20
0
500
1000
1500
2000
2500
3000
3500
GrLivArea



Bivariate

இரண்டு variables எவ்வாறு தொடர்பு கொண்டுள்ளன என வரைபடம் வரைந்து பார்ப்பது bi-variate analysis ஆகும். இதன் X-அச்சில் ஒன்றும் Y-அச்சில் மற்றொன்றும் வைத்து வரைபடம் வரையப்படும்.

இங்கு ஒவ்வொரு வீட்டினுடைய sqft அளவைப் பொறுத்து அதன் விற்பனை விலை எவ்வாறு மாறுபடுகிறது என்பது scatter plot, heatmap ஆகியவை மூலம் காட்டப்பட்டுள்ளன. HeatMap-ல் இரண்டு வரைபடங்கள் உள்ளன. ஒன்று seaborn வழங்குகின்ற வரைபடமாகவும், மற்றொன்று matplotlib வழங்குகின்ற வரைபடமாகவும் உள்ளது.

Scatter plot என்பது தரவுகள் இருக்கும் இடத்தை தனித்தனி புள்ளிகளாகக் காட்டும். இதில் தரவுகளைக் குறிப்பிடுவதற்கு புள்ளிகளுக்கு பதிலாக, சிறுசிறு வட்டங்களையோ அல்லது வேறு சில வடிவங்களையோ கூட பயன்படுத்தலாம்.

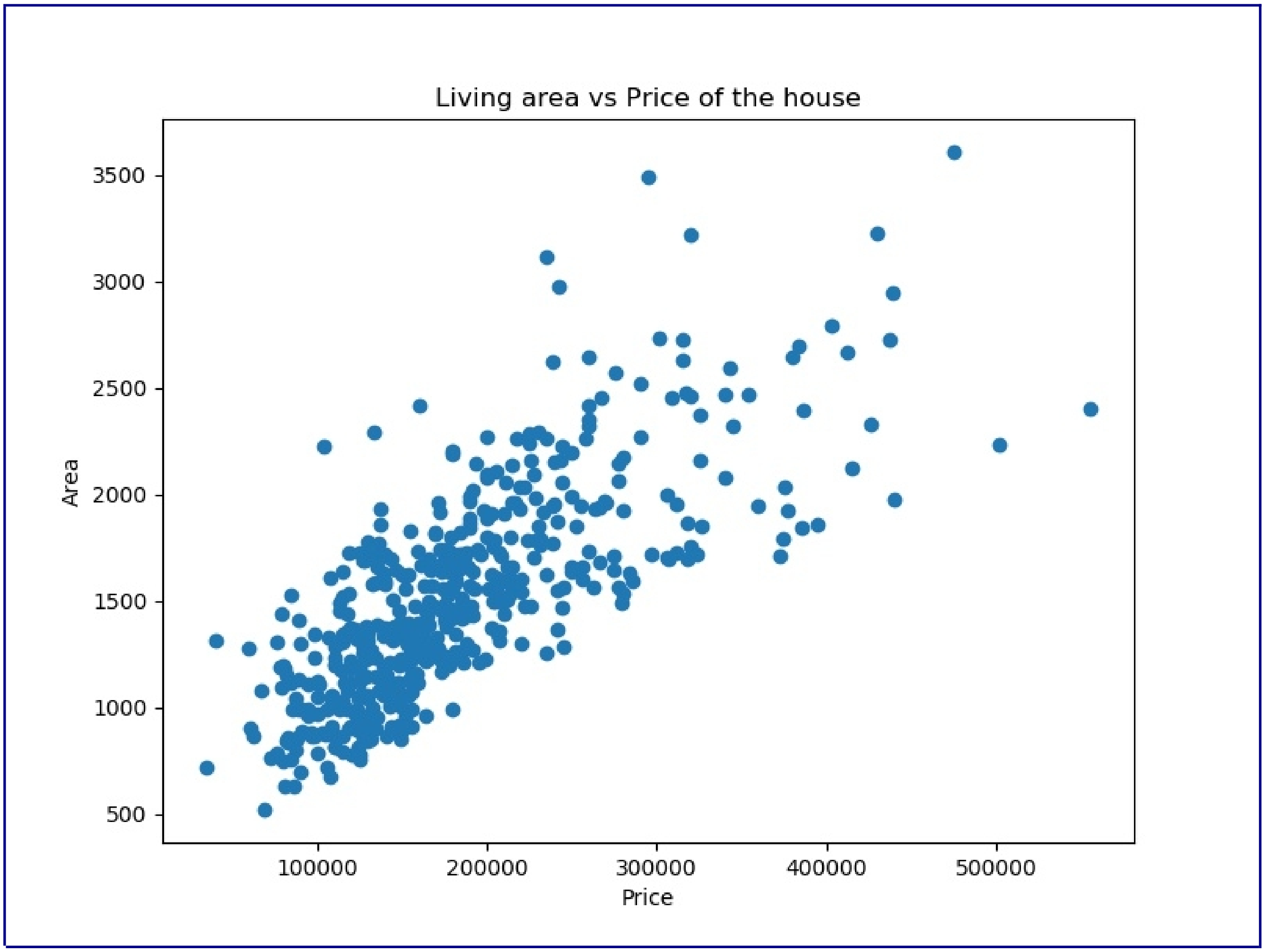
*Heatmap என்பது 2 dimensional data-வை வரைந்து காட்ட உதவும் வரைபட வகை ஆகும். இங்கு 12*12*

மதிப்பு கொண்ட வரைபடம் வரையப்பட்டுள்ளது. Matrix-ல் உள்ள ஒவ்வொரு தனித்தனி மதிப்பும் தனித்தனி நிறத்தால் குறிக்கப்படும். இது பொதுவாக நமது தரவுகள் எவ்விதத்தில் அமைந்துள்ளன எனக் காண உதவும். seaborn மற்றும் matplotlib வழங்குகின்ற இரண்டு வகையான heatmaps இங்கு கொடுக்கப்பட்டுள்ளன.

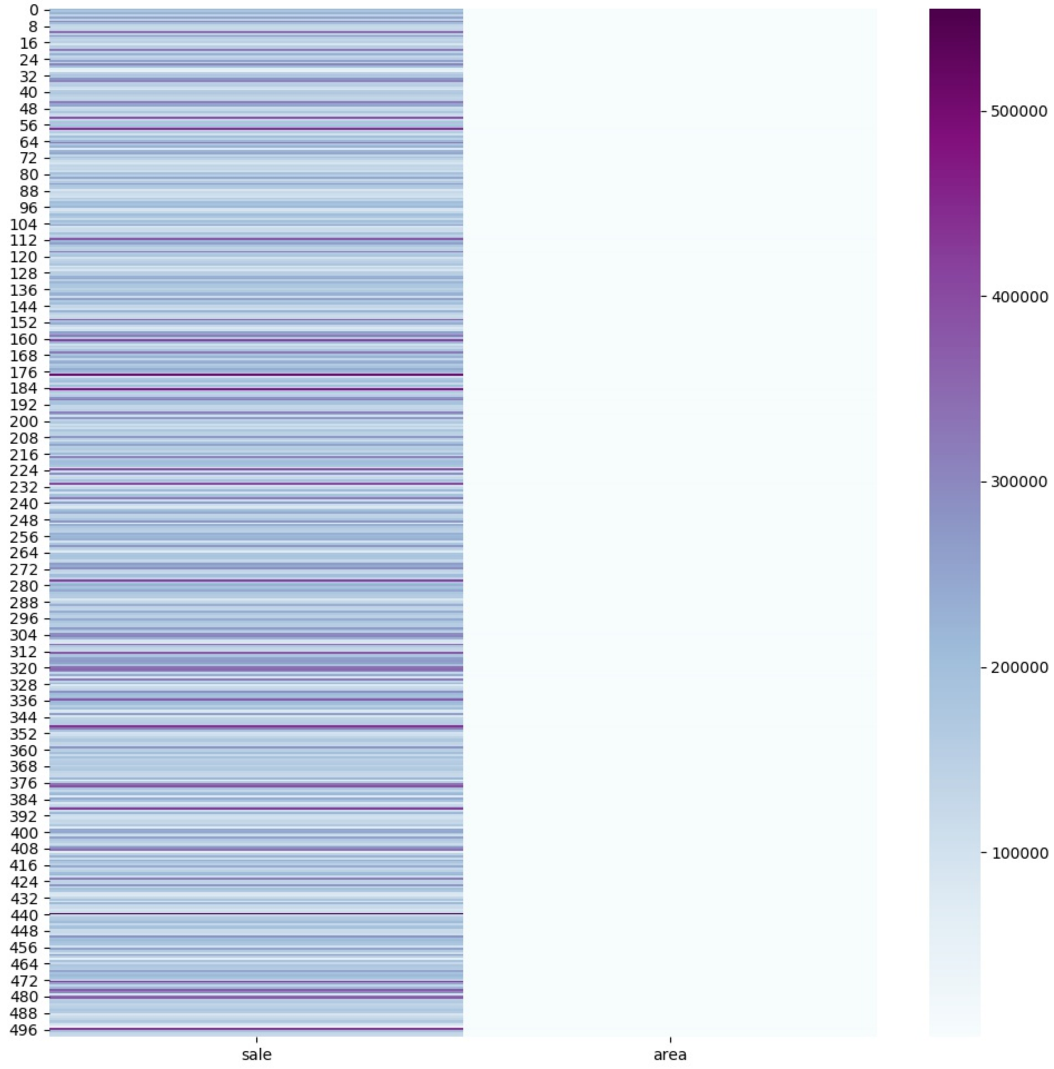
https://gist.github.com/nithyadurai87/d93a853d86cf5500011cb41308dd1935

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns


df = pd.read_csv("14_input_data.csv")
df = df.fillna(0)
df = df[:500]

fig = plt.figure(figsize=(8,6))
ax = fig.add_subplot(111)
ax.set(title='Living area vs Price of the house',
      xlabel='Price', ylabel='Area')
price = df['SalePrice'].tolist()
area = df['GrLivArea'].tolist()
ax.scatter(price,area)
plt.savefig('ScatterPlot.jpg')

df2 = pd.DataFrame()
df2['sale'] = df['SalePrice']
df2['area'] = df['GrLivArea']
fig = plt.figure(figsize=(12,12))
r = sns.heatmap(df2, cmap='BuPu')
plt.savefig('HeatMapSeaborn.jpg')

fig = plt.figure(figsize=(8,6))
ax = fig.add_subplot(111)
ax.set(title="Total Living Sq.Ft",
      ylabel='No of Houses', xlabel='Living Sq.Ft')
ax.hist2d(price,area,bins=100)
plt.savefig('HeatMapMatplotlib.jpg')
```

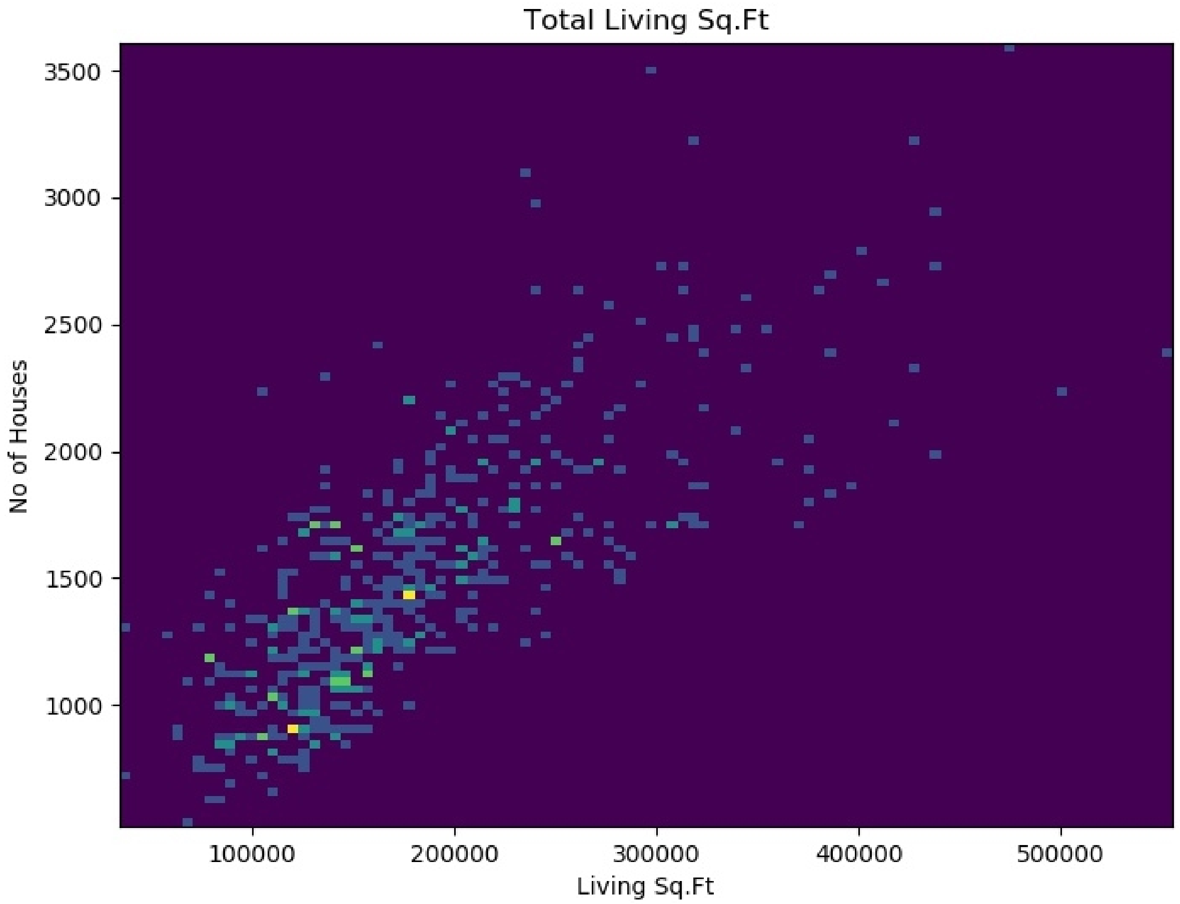
Scatter Plot



HeatMap - Seaborn



HeatMap - Matplotlib



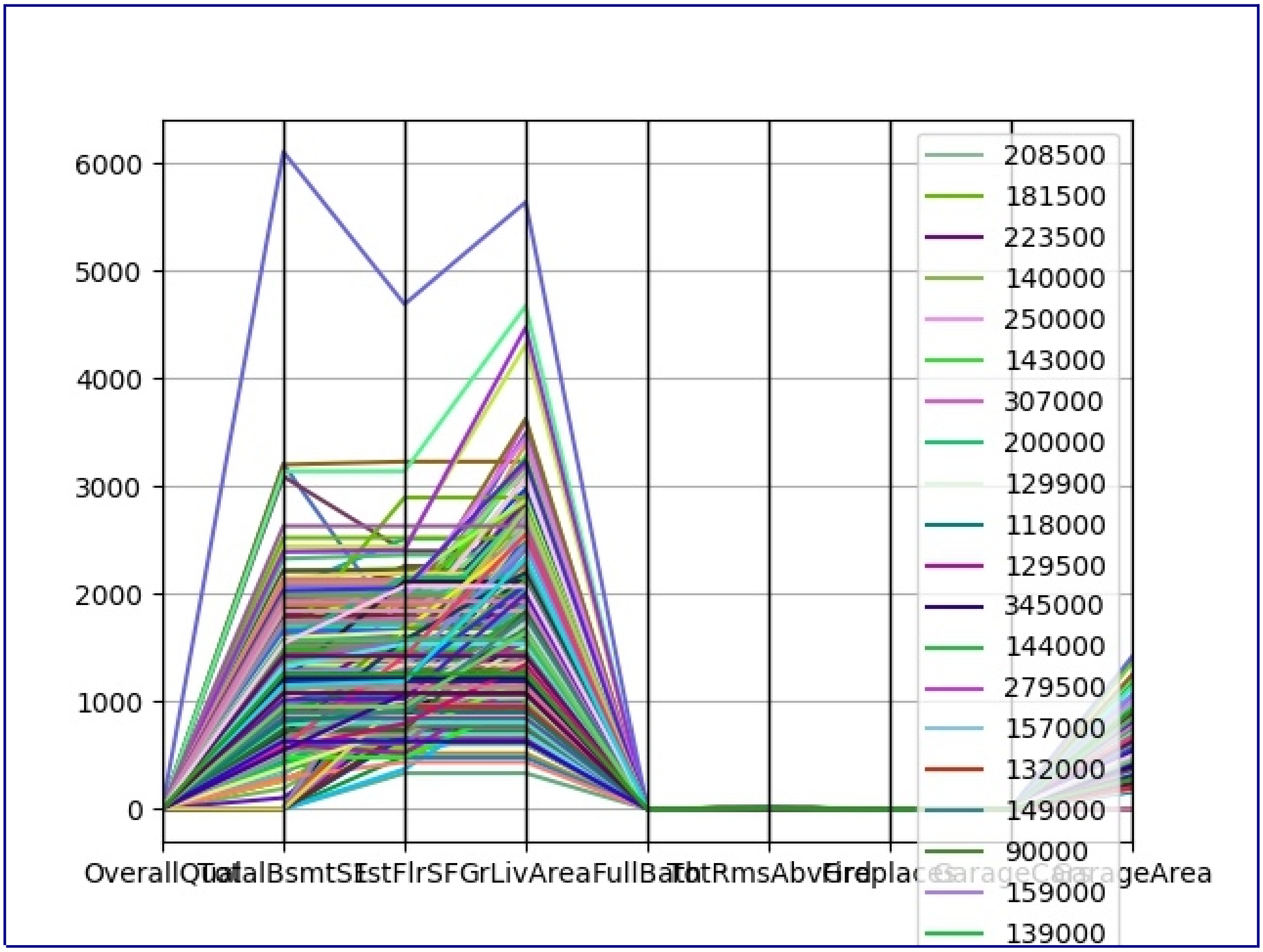
Multivariate

இரண்டுக்கும் மேற்பட்ட மதிப்புகளைப் பொறுத்து ஒரு *taraget variable* எவ்வாறு அமைகிறது எனக் காண்பதே *multi-variate analysis* ஆகும். *Parallel coordinates* என்பது இத்தகைய *multi dimensional data*-வைக் காண்பதற்கு உதவும் வரைபட வகை ஆகும்.

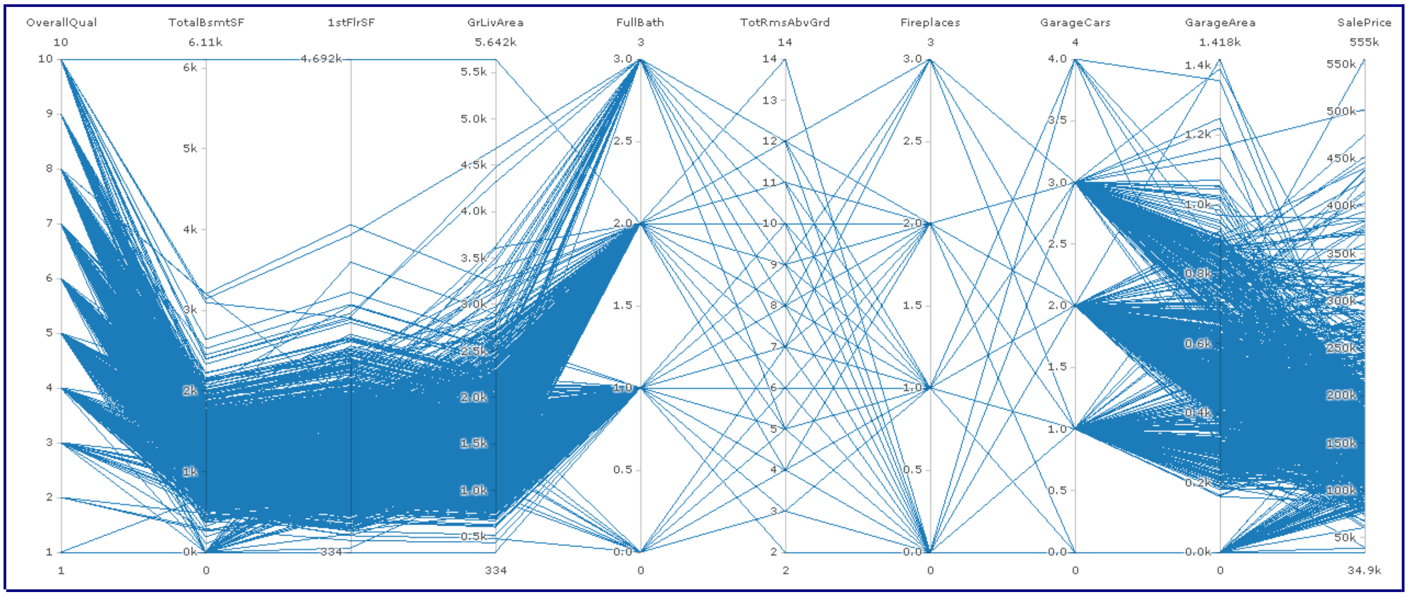
இங்கு *plotly* மற்றும் *matplotlib* மூலம் இத்தகைய வரைபடங்கள் வரைந்து கட்டப்பட்டுள்ளது. *'SalePrice'* எனும் *categorical variable*-க்கு தரவுகள் எவ்வாறு சீராகப் பரவியுள்ளது என்பதை இந்த வரைபடம் காட்டும். இதை வைத்து இதில் ஏதாவது *trend* உள்ளதா என்பதை நாம் கண்டறியலாம். *Plotly* மூலம் வரையும் போது, ஒவ்வொரு *column*-லும் உள்ள *min* மற்றும் *max* மதிப்புகளை அதன் *range*-ஆக கொடுக்கப்பட்டுள்ளதை கவனிக்கவும். இந்த வரைபடம் ஒரு *html* கோப்பாக *interactive* முறையில் சேமிக்கப்படுகிறது.

[*https://gist.github.com/nithyadurai87/2b0bb469694d33c7d1472880f10f67e1*](https://gist.github.com/nithyadurai87/2b0bb469694d33c7d1472880f10f67e1)

```
import pandas as pd
import matplotlib.pyplot as plt
from pandas.plotting import parallel_coordinates
import plotly
import plotly.graph_objs as go
import numpy as np

df = pd.read_csv("14_input_data.csv")
parallel_coordinates(df, 'SalePrice')
plt.savefig('ParallelCoordinates.jpg')

desc_data = df.describe()
desc_data.to_csv('./metrics.csv')

X = df[list(df.columns)[:-1]]
y = df['SalePrice']

data = [
    go.Parcoords(
        line = dict(colorscale = 'Jet',
                   showscale = True,
                   reversescale = True,
                   cmin = -4000,
                   cmax = -100),
        dimensions = list([
            dict(range = [1,10],
                 label = 'OverallQual', values = df['OverallQual']),
            dict(range = [0,6110],
                 label = 'TotalBsmtSF', values = df['TotalBsmtSF']),
            dict(tickvals = [334,4692],
                 label = '1stFlrSF', values = df['1stFlrSF']),
            dict(range = [334,5642],
                 label = 'GrLivArea', values = df['GrLivArea']),
            dict(range = [0,3],
                 label = 'FullBath', values = df['FullBath']),
            dict(range = [2,14],
                 label = 'TotRmsAbvGrd', values = df['TotRmsAbvGrd']),
            dict(range = [0,3],
                 label = 'Fireplaces', values = df['Fireplaces']),
            dict(range = [0,4],
                 label = 'GarageCars', values = df['GarageCars']),
            dict(range = [0,1418],
                 label = 'GarageArea', values = df['GarageArea']),
            dict(range = [34900,555000],
                 label = 'SalePrice', values = df['SalePrice'])
        ])
    )
]
plotly.offline.plot(data, filename = './parallel_coordinates_plot.html',
auto_open= True)
```



6000
5000
4000
3000
2000
1000
0
OverallQual
TotalBsmtSF
1stFlrSF
GrLivArea
FullBath
TotRmsAbvGrd
Fireplaces
GarageCars
GarageArea
208500
181500
223500
140000
250000
143000
307000
200000
129900
118000
129500
345000
144000
279500
157000
132000
149000
90000
159000
139000



Polynomial Regression

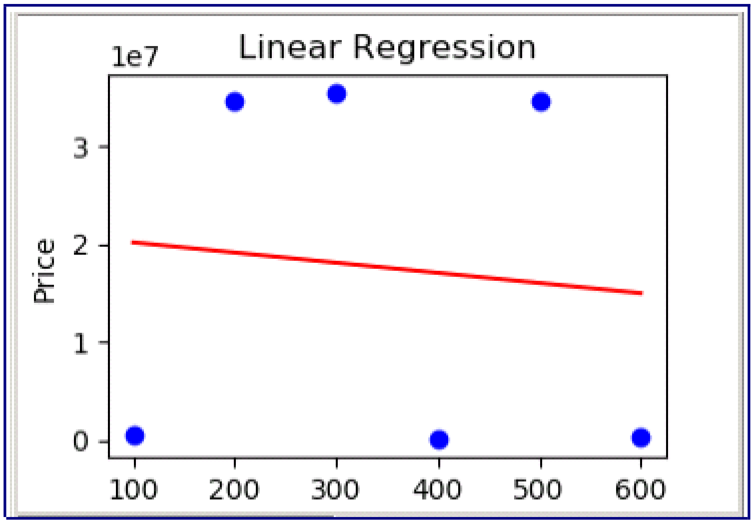
ஒரு நேர் கோட்டில் பொருந்தாத சற்று சிக்கலான தரவுகளுக்கு polynomial regression-ஐப் பயன்படுத்தலாம். கீழ்க்கண்ட நிரலில் ஒரு வீட்டிற்கான சதுர அடியும், அதற்கான விலையும் கொடுக்கப்பட்டுள்ளது. இதில் linear மற்றும் 2nd order, 3rd order, 4th order & 5th order polynomial பொருத்திப் பார்க்கப் படுகிறது.

https://gist.github.com/nithyadurai87/b7d3bf7733b5d4a8d2c8b2d1b8dcb531

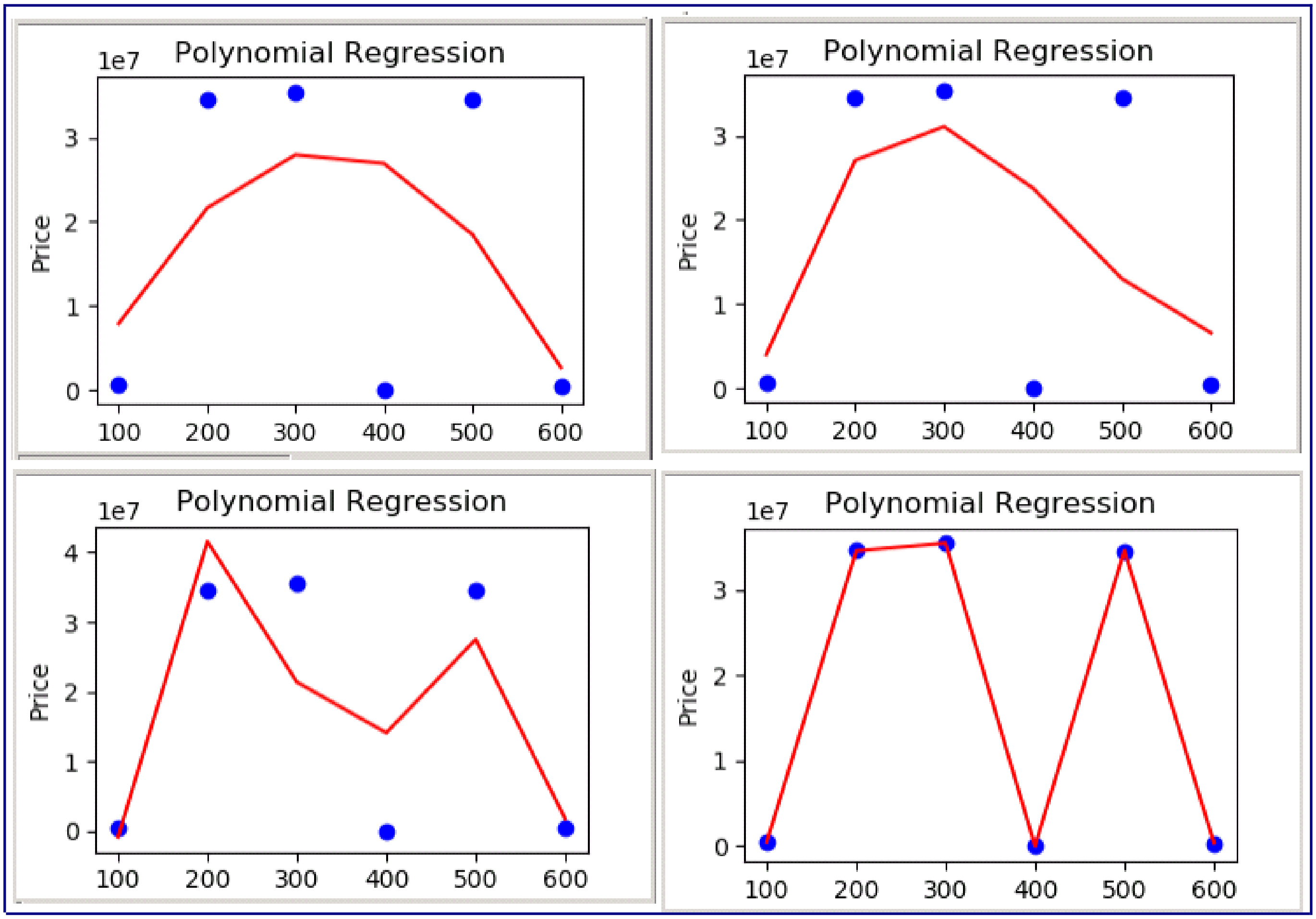
```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures

X = pd.DataFrame([100,200,300,400,500,600],columns=['sqft'])
y = 
pd.DataFrame([543543,34543543,35435345,34534,34534534,345345],columns=['Price'])

lin = LinearRegression()
lin.fit(X, y)
plt.scatter(X, y, color = 'blue')
plt.plot(X, lin.predict(X), color = 'red')
plt.title('Linear Regression')
plt.xlabel('sqft')
plt.ylabel('Price')
plt.show()

for i in [2,3,4,5]:
      poly = PolynomialFeatures(degree = i)
      X_poly = poly.fit_transform(X)
      poly.fit(X_poly, y)
```

```
      lin2 = LinearRegression()
      lin2.fit(X_poly, y)
      plt.scatter(X, y, color = 'blue')
      plt.plot(X, lin2.predict(poly.fit_transform(X)), color = 'red')
      plt.title('Polynomial Regression')
      plt.xlabel('sqft')
      plt.ylabel('Price')
plt.show()
```

linear regression-ஐ வைத்துப் பொருத்தும் போது, அதற்கான கோடு எந்த ஒரு தரவுகளின் மீதும் பொருந்தாமல் பின்வருமாறு அமைகிறது. இதுவே under fitting எனப்படும்.



எனவே 2nd order முறையில் அதனுடைய cube கண்டுபிடிக்கப்பட்டு அவற்றை தரவுகளுடன் பொருத்த முயலும்போது பின்வருமாறு அமைகிறது. இதுவே non-linear function எனப்படும். அதாவது இது ஒரு நேர் கோடாக அமையாது.



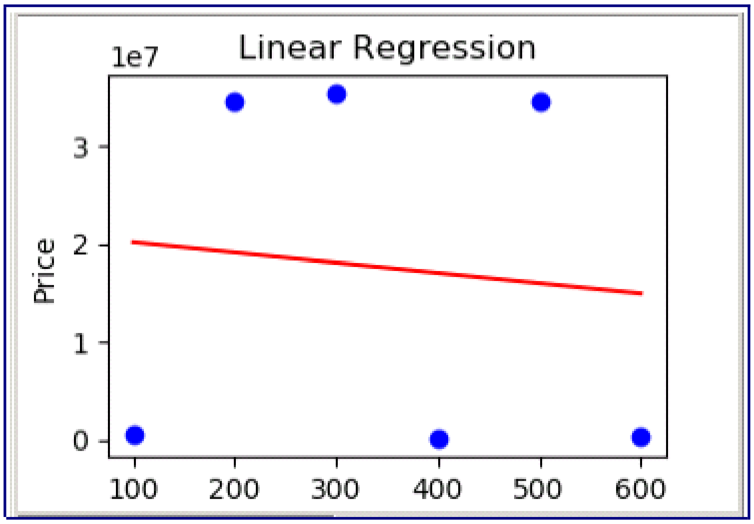
அவ்வாறே 3rd *order*-ல் தரவுகளுடைய *cube* கண்டுபிடிக்கப்பட்டு அவை தரவுகளுக்கு இன்னும் சற்று அருகில் செல்வதைக் காணலாம்.

கடைசியாக 4th *order*-ல் அனைத்துத் தரவுகளின் மீதும் முழுதாகப் பொருந்துமாறு *non-linear* அமைகிறது. இதுவே *over fitting* என்று அழைக்கப்படும். இதுபோன்ற *over fitting*-ம் சரியானது அல்ல.

எனவே எந்த *order*-ல் அனைத்துத் தரவுகளின் மீதும், நமது *non-linear* பரவலாகப் பொருந்துகிறதோ (*over fitting* அல்லாமல்), அதையே நாம் கணிப்பிற்கு எடுத்துக் கொள்ளலாம். இம்முறையில் ஒரு எண்ணிற்கு அடுத்தடுத்த மடங்குகள் கண்டுபிடிக்கப்படுவதால் , இதற்கான சமன்பாடு அதன் மடங்குகளைப் பொறுத்து பின்வருமாறு அமைகிறது. அதிக அளவில் எண்கள் அதிகரிக்கப்படுவதால் *feature scaling*-ன் பயன்பாடு இங்கு அதிக முக்கியத்துவம் பெறுகிறது.

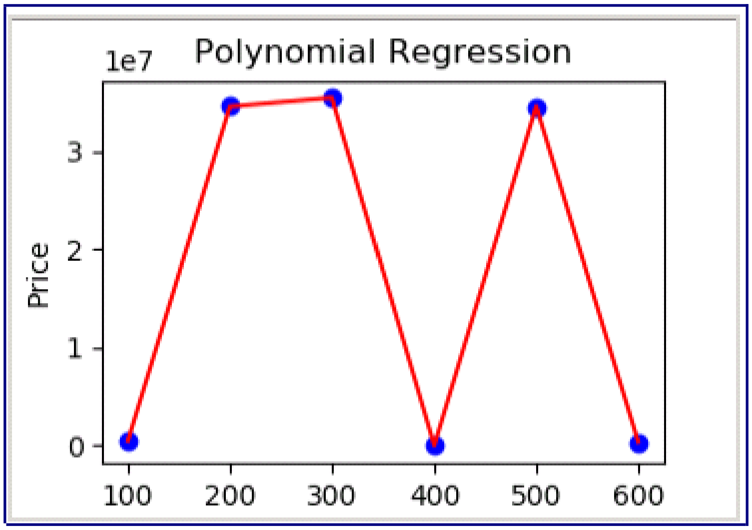
$$h(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 \ldots$$

Underfitting – High bias



கணிப்புக்கான கோடானது தரவுகளின் மீது அதிகமாகப் பொருந்தாத நிலையே *underfitting* எனப்படுகிறது. அதிக அளவு தரவுகளுக்கு குறைந்த *features* கொண்டு கணிக்கும் போது இந்நிலை ஏற்படுகிறது. இதுவே *high bias* பிரச்சனை என்றும் அழைக்கப்படுகிறது. ஏனெனில் மிகக் குறைந்த அளவு அம்சங்களைச் சார்ந்தே இது செயல்படுகிறது. உதாரணத்துக்கு *50,000* தரவுகளுக்கு*(m)* இரண்டே இரண்டு *features*-ஐக் கொண்டு கணிக்கும் போது தரவுகள் எதுவும் கோட்டில் பொருந்தாது. எனவே இதுபோன்ற பிரச்சனைக்கு தரவுகளின் எண்ணிக்கையை அதிகரிப்பது தீர்வாகாது. *features*- ன் எண்ணிக்கையை மட்டுமே அதிகரிக்க வேண்டும். .

Overfitting – High variance



அதிக அளவு *features*-ஐ சேர்ப்பதன் மூலம் *underfitting*-ஐத் தவிர்க்கலாம் என ஏற்கெனவே பார்த்தோம். அதுவே அளவுக்கு அதிகமாக சேர்த்துவிட்டால், *overfitting* என்ற நிலை ஏற்பட்டு விடுகிறது. இதனைத் தவிர்ப்பதற்காக சேர்க்கப்படுவதே *regularization parameter* ஆகும். அதாவது தரவுகளின் எண்ணிக்கை

குறைவாக இருந்து, features அதிகமாக இருக்கும்போது இந்நிலை ஏற்படும். உதாரணத்துக்கு வெறும் 50 தரவுகளுக்கு, 250 features கொண்டு கணிக்கும்போது கோடானது, அனைத்துத் தரவுகளின் மீதும் அளவுக்கு அதிகமாகப் பொருந்துகிறது. இதுவே high variance என்று அழைக்கப்படுகிறது. இதனைத் தவிர்க்க features எண்ணிக்கையை மிகவும் குறைத்தாலும் high bias ஆகிவிடுகிறது. இதுவே bias-variance tradeoff என்று அழைக்கப்படுகிறது. இது போன்ற பிரச்சனைகளை தவிர்க்க features எண்ணிக்கையை சரியான அளவுக்கு குறைக்க வேண்டும் அல்லது regularization-ஐப் பயன்படுத்தலாம்.

Regularization

இது ஒவ்வொரு feature-வுடனும் இணைக்கப்படும் parameter-ன் (தீட்டாக்களின்) அளவைக் குறைக்கிறது. எனவே features-ன் எண்ணிக்கை அதிகமாக இருந்தாலும், அவை கணிப்பில் குறைந்த அளவே பங்கேற்குமாறு செய்யலாம். linear regression-வுடன் இது இணையும் போது, அதற்கான சமன்பாடு பின்வருமாறு அமைகிறது.

Linear regression:

$$J = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2 + \lambda \sum_{j=1}^{n} \theta_j^2$$

இதில் லாம்டா என்பதுதான் regularization-க்கான parameter. இதன் மதிப்பு 1 லிருந்து தொடங்கி அனைத்து feature -க்கும் அமைவதைக் காணவும் (j =1 to n). ஏனெனில் x0 -ன் மதிப்பு எப்போதும் 1 என இருக்குமென்பதை ஏற்கனவே கண்டோம். ஆகவே தீட்டா0 -வுடைய மதிப்பைக் குறைக்கத் தேவையில்லை.

அதேபோல் லாம்டாவின் மதிப்பு மிக அதிகமாகவும் இருக்கக் கூடாது. மிகக் குறைவாகவும் இருக்கக் கூடாது. குறைவாக இருந்தாலும், overfitting-ஐத் தவிர்க்காது. அதிகமாக இருந்தாலும் bias ஏற்படக் காரணமாகிவிடும். எனவே சரியான அளவில் இருக்க வேண்டும்.

Gradient descent-வுடன் regularization இணையும்போது, அதற்கான சமன்பாடு பின்வருமாறு அமையும். இங்கும் தீட்டா0 -வுடன் இணையாமல், தீட்டா1 -லிருந்து regularization இணைக்கப்படுகிறது.

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h(x) - y\right)$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h(x) - y\right) \cdot x + \frac{\lambda}{m} \theta_1$$

$$:= \theta_1 \left(1 - \alpha \frac{\lambda}{m}\right) - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h(x) - y\right) \cdot x$$

குறைந்த cost கண்டுபிடிப்பதற்கான சாதாரண சூத்திரத்துடன் regularization இணையும்போது, அது

பின்வருமாறு அமையும்.

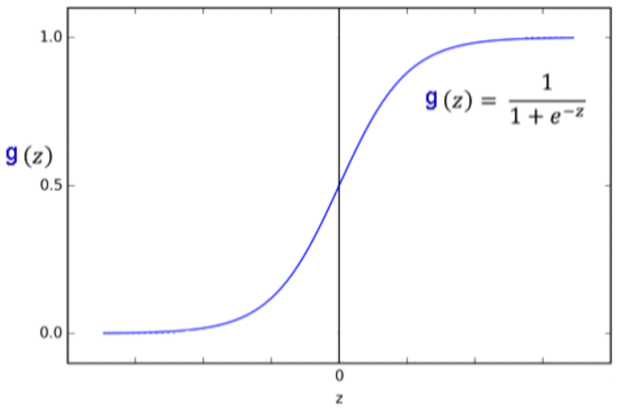
Normal Equation:

$$\theta = \left(X^T X + \lambda \begin{bmatrix} 0 & & & \\ & 1 & & \\ & & 1 & \\ & & & \ddots & \\ & & & & 1 \end{bmatrix} \right)^{-1} X^T y$$

Logistic regression

நமது கணிப்பு ஒரு முழு மதிப்பினை வெளிப்படுத்தாமல், ஏதேனும் ஒரு வகையின் கீழ் அமைந்தால், அதுவே logistic regression எனப்படும். இந்த வகைப்படுத்தல், binary மற்றும் multiclass எனும் இரு விதங்களில் நடைபெறும். logistic regression என்பது இதற்கு உதவுகின்ற ஒரு algorithm ஆகும். இதன் பெயரில் மட்டும்தான் regression எனும் வார்த்தை உள்ளது. ஆனால் இது ஒரு classification-க்கான algorithm ஆகும்.

$$\begin{aligned} h(x) &= 0 \quad \text{or} \quad 1 \\ &= 0 \leq h(x) \leq 1 \\ &= g(z) \\ &= g(\theta_0 x_0 + \theta_1 x_1 + \cdot\cdot + \theta_n x_n) \\ &= g(\theta^T x) \\ &= \frac{1}{1 + e^{-\theta^T x}} \end{aligned}$$



Sigmoid function

ஒரு விஷயம் நடைபெறுமா? நடைபெறாதா? அல்லது இருக்கா? இல்லையா? என்பதையே இது கணிக்கிறது. ஆம் என்பது 1 எனவும் இல்லை என்பது 0 எனவும் கணிக்கப்படும். ஆகவே இதன் கணிப்பானது 0-லிருந்து 1-வரை அமையும். இதற்கான வரைபடம் பின்வருமாறு. அந்த வரைபடத்தில் z-ன் மதிப்பைப் பொறுத்து கணிக்கப்படும் g(z), 0-முதல் 1-வரை அமைய வேண்டுமெனில் அதற்கான சூத்திரமானது 1/(1+e**-z) என்று அமையும். இதுவே sigmoid function என்று அழைக்கப்படுகிறது.

எனவே z-க்கான இடத்தில் h(x) -ஐப் பொருத்தினால், அது 0-1 வரை அமைவதற்கான சமன்பாடாக பின்வரும் சூத்திரம் அமையும். இதுவே logictic regression-க்கான சமன்பாடு ஆகும்.

ஒரு மின்னஞ்சல் spam-ஆ இல்லையா எனக் கணிப்பதற்கான நிரல் பின்வருமாறு.

https://gist.github.com/nithyadurai87/f09984303f976ca6eb8a64a4b7f0e391

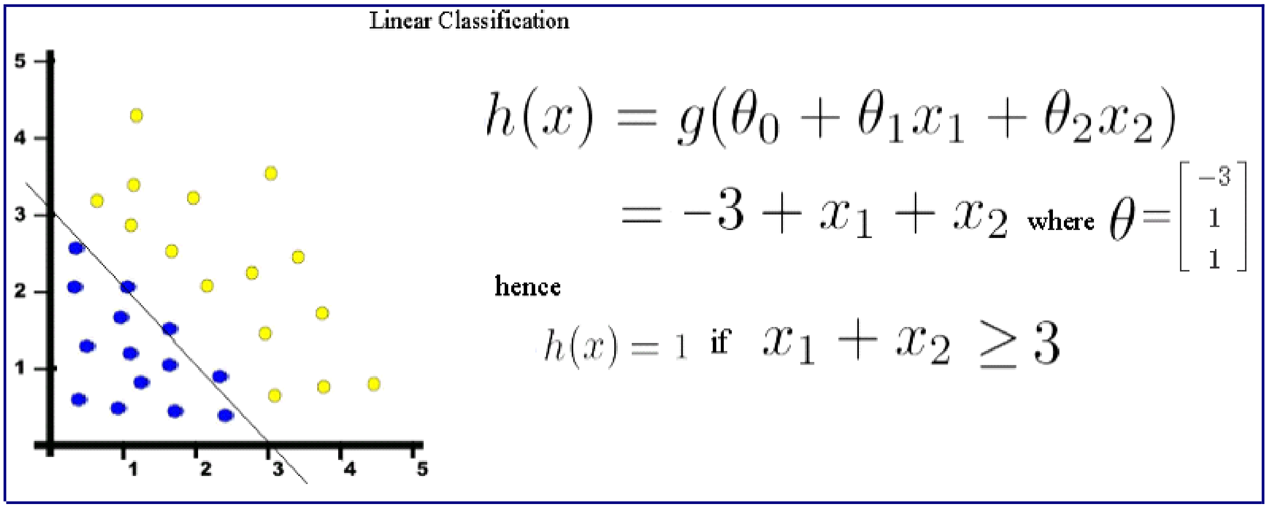
```
import numpy as np
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model.logistic import LogisticRegression
from sklearn.model_selection import train_test_split, cross_val_score


df = pd.read_csv('./spam.csv', delimiter=',',header=None)
X_train_raw, X_test_raw, y_train, y_test = train_test_split(df[1],df[0])

vectorizer = TfidfVectorizer()
X_train = vectorizer.fit_transform(X_train_raw)
X_test = vectorizer.transform(X_test_raw)
classifier = LogisticRegression()
classifier.fit(X_train, y_train)
predictions = classifier.predict(X_test)
print(predictions)
```

['ham' 'ham' 'ham']

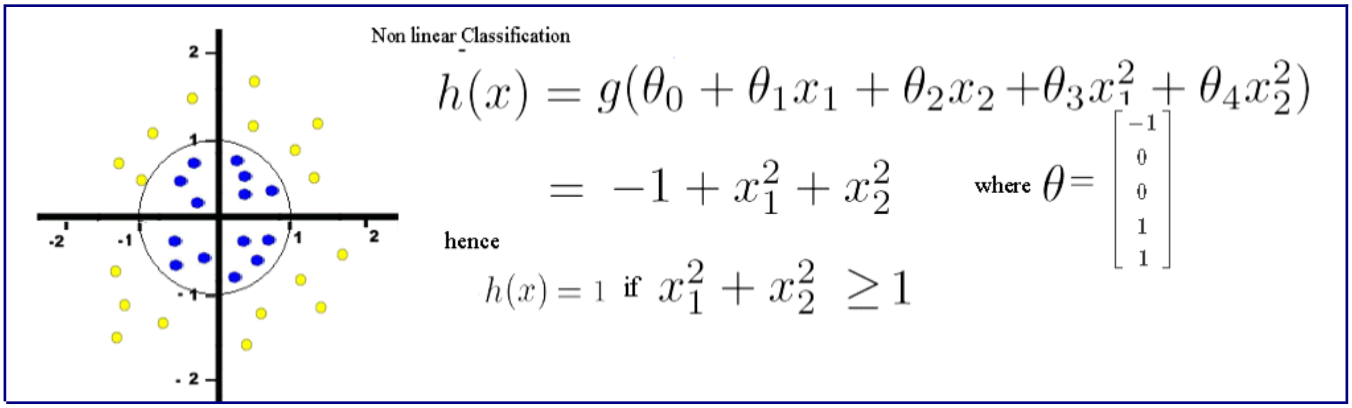
Decision Boundary

h(x) = 1 என்பது எப்போதும் ஆம் என்பதையே குறிக்கும். எனவே 1-h(x) என்பது இல்லை என்பதைக் குறிக்கும். உதாரணத்துக்கு h(x) என்பது நாளை மழை பெய்ய 70% வாய்ப்பு உள்ளது என கணிக்கிறதெனில், மீதமுள்ள 30% இல்லை என்பதைக் கணித்துள்ளது என்றே அர்த்தம்.

தரவுகள் கீழ்க்கண்ட வரைபடத்தில் காணப்படுவதுபோல் பரவலாக அமைந்திருக்கிறது எனில், எதற்கு மேல் சென்றால் ஆம் எனக் கணிக்கலாம், எதற்குக் கீழ் அமைந்தால் இல்லை எனக் கணிக்கலாம் என்பதை முடிவு செய்வதே decision boundary ஆகும். இது எப்போதும் தீட்டா மதிப்புகளைப் பொறுத்தே அமையும். -3, 1, 1 எனும் மதிப்புகளை தீட்டா0, தீட்டா1, தீட்டா2 எனுமிடத்தில் பொருத்தினால், h(x)=1 என கணிப்பதற்கு x1 மற்றும் x2-ஆனது 3-க்கு மேல் அமைய வேண்டும் என்பதை decision boundary-ஆக அமைத்துள்ளது.

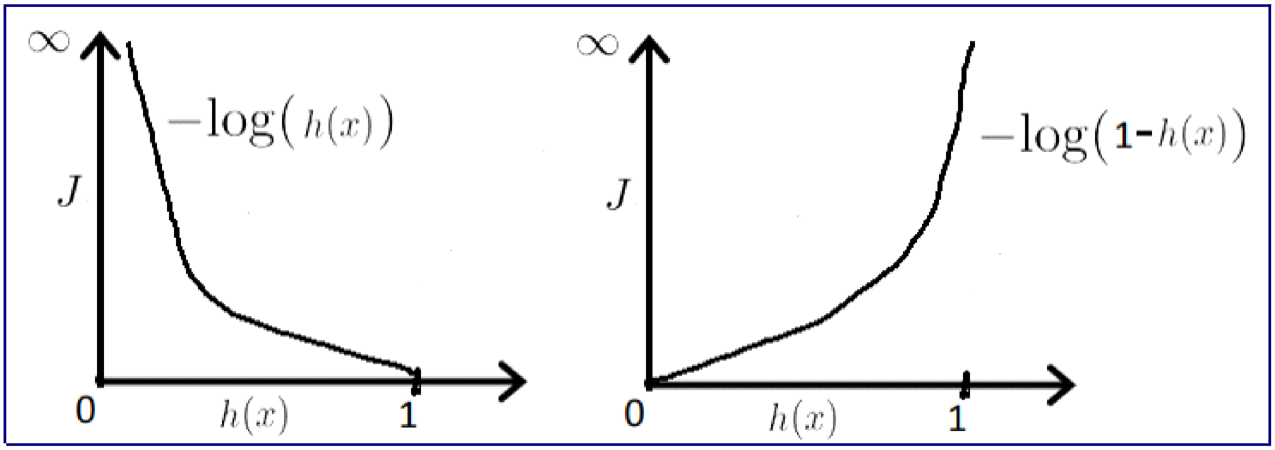


தரவுகள் கீழ்க்கண்டவாறு *non-linear* முறையில் பரவியிருப்பதால், இதன் தீட்டா மதிப்புகளான *-1,0,0,1,1* என்பது *2*-ம் *order polynomial*-ல் இருக்கும் சமன்பாட்டில் பொருத்தப்படுகிறது. *1*-என்பது *boundary*-ஆக கண்டு பிடிக்கப்பட்டுள்ளது. இதுவே *threshold classifier* என்றும் அழைக்கப்படும்.



Cost function

உண்மையில் நாளை மழை பெய்ய வாய்ப்பு இருக்கிறது என கணிக்கப்பட வேண்டியது இல்லை என கணிக்கப்பட்டால், அது ஒரு *error*. அவ்வாறே பெய்யாது என்பதை பெய்யும் எனக் கணித்தாலும் அது ஒரு *error*. அதாவது *1* என்பது *0* என கணிக்கப்பட்டாலோ அல்லது *0* என்பது *1* என கணிக்கப்பட்டாலோ அதனுடைய தவறு எத்தனை சதவீதம் நிகழ்ந்துள்ளது என்பதைக் கணக்கிட்டுக் கூற இயலாது. *Infinity* (எண்ணற்ற) என்பதே அதன் மதிப்பாக இருக்கும். இதற்கான வரைபடங்கள் பின்வருமாறு. அதில் *x* என்பது *h(x)* எனில், *y* -ஆனது *infinity*-ஐ நோக்கிச் செல்லும் வளைவுக்கான சூத்திரம் *-log(h(x))*



அதாவது,

1 என்பது 0 என கணிக்கப்பட்டால் அதற்கான cost = -log(h(x)), அவ்வாறே

0 என்பது 1 என கணிக்கப்பட்டால் அதற்கான cost = -log(1-h(x))

எனவே cost-க்கான சூத்திரம் பின்வருமாறு அமைகிறது. இதில் y=1 எனவும் y=0 எனவும் வைத்து சரிபார்த்துக் கொள்ளவும்.

$$J= -\frac{1}{m}[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1-y^{(i)}) \log (1-h_\theta(x^{(i)}))]$$

When y = 1,

= y . log(h(x)) + (1-y).log(1-h(x))

= 1.log(h(x)) + (1-1).log(1-h(x))

= log(h(x)) + 0

= log(h(x))

When y = 0,

= y . log(h(x)) + (1-y).log(1-h(x))

= 0.log(h(x)) + (1-0).log(1-h(x))

= 0 + 1.log(1-h(x))

= log(1-h(x))

இதற்கான contour plots ஒரே ஒரு கிண்ண வடிவ அமைப்பில் அமையாமல், சிறு சிறு வளைவுகளைப் பெற்று பல்வேறு ஏற்ற இறக்கங்களைக் கொண்டிருக்கும். இதுவே non-convex function எனப்படும். அதாவது regression-க்கான வரைபடத்தில் ஒரே ஒரு global optimum மட்டும் காணப்படும். ஆனால் classification-க்கான வரைபடத்தில் பல்வேறு local optimum காணப்படும். ஏனெனில் இங்கு 'இருக்கு', 'இல்லை' எனும் இரண்டு மதிப்புகள் மட்டும் மாறி மாறி கணிக்கப்படுவதால், பல்வேறு local optimums இருக்கின்றன. இது போன்ற non-convex function-லும் நாம் gradient descent-ஐப் பயன்படுத்தலாம்.

இதற்கான gradient descent-ன் சமன்பாடும் multiple linear-ஐ ஒத்தே இருக்கும். ஒரே ஒரு வித்தியாசம் என்னவெனில், h(x) -க்கான தீட்டா-transpose.x என்பது இங்கு sigmoid function-ஐக் கொண்டிருக்கும்.

Classification accuracy

நாளை உண்மையிலேயே மழை பெய்ய வாய்ப்பு இருக்கும்போது 'இல்லை' எனக் கணிப்பதும், இல்லாதபோது 'இருக்கு' எனக் கணிப்பதும் classification-ல் நடைபெறும் தவறு ஆகும். எனவே எவ்வளவு தரவுகளுக்கு சரியான கணிப்புகள் நிகழ்ந்துள்ளது எனக் கண்டறிவதே accuracy ஆகும்.

ஒரு பத்து நாளைக்கான வானிலை கணிப்புகள் கீழ்க்கண்ட உதாரணத்தில் காணப்படுவதுபோல் இருக்கிறது என வைத்துக் கொள்வோம். அதாவது y_true -ல் உண்மையிலேயே மழை பெய்ததா, இல்லையா எனும் விவரம் 1 மற்றும் 0 ஆக உள்ளது. அதற்கான கணிப்புகள் y_pred -ல் உள்ளது. இவற்றை ஒப்பிட்டுப் பார்க்கும்போது இரண்டாவது, ஆறாவது மற்றும் ஏழாவது கணிப்புகள் மட்டும் மாறி நடைபெற்றிருப்பதை கவனிக்கவும். எனவே மொத்த 10 தரவுகளில், 3 மட்டும் தவறாக அமைந்திருப்பதால், இதன் accuracy 70% என வந்துள்ளது.

https://gist.github.com/nithyadurai87/7668ce262ed9070d89b158bb7f13c5cb

```
from sklearn.metrics import precision_recall_fscore_support
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt

y_true = [0, 0, 0, 0, 0, 1, 1, 1, 1, 1]
y_pred = [0, 1, 0, 0, 0, 0, 0, 1, 1, 1]

print ('Accuracy:', accuracy_score(y_true, y_pred))
print (confusion_matrix(y_true, y_pred))
print (precision_recall_fscore_support(y_true, y_pred))

plt.matshow(confusion_matrix(y_true, y_pred))
plt.title('Confusion matrix')
plt.colorbar()
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```

Accuracy: 0.7
[[4 1]
[2 3]]
(array([0.66666667, 0.75]), array([0.8,0.6]), array([0.72727273, 0.666666
67]), array([5,5], dtype=int64))

Confusion Matrix

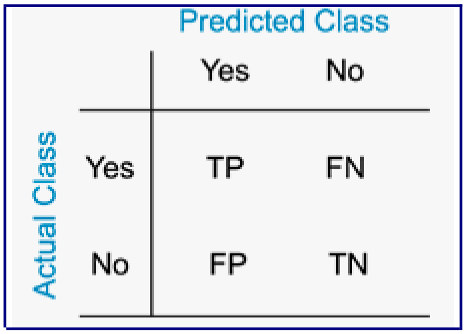
இது பின்வரும் விதிகளின் படி உருவாக்கப்படுகிறது.

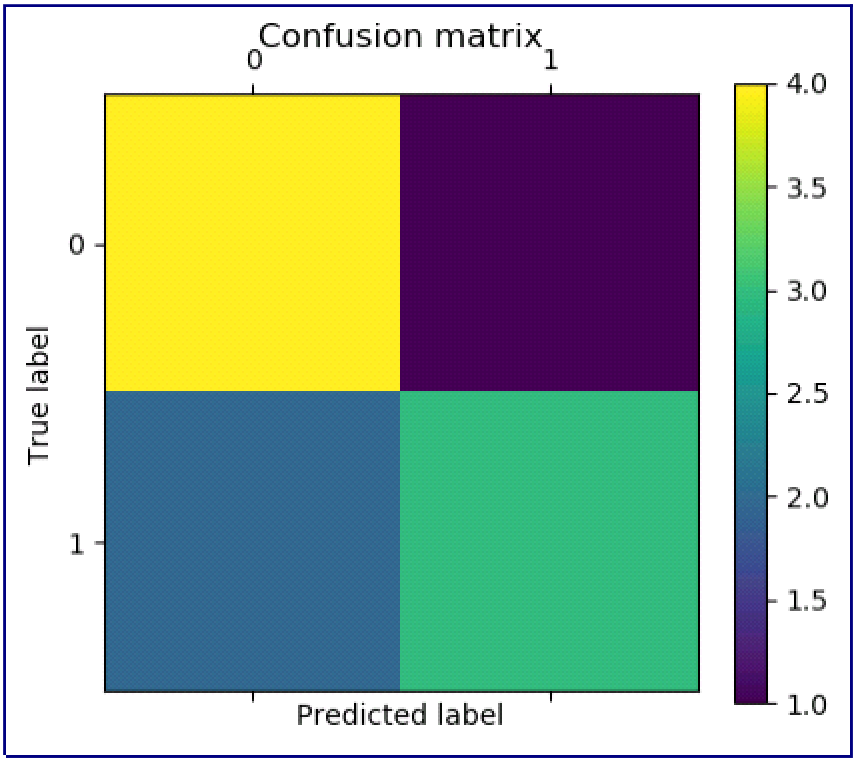
0 எனும் மதிப்பு 1 என கணிக்கப்பட்டால் = False Positive

1 எனும் மதிப்பு 0 என கணிக்கப்பட்டால் = False Negative

1 எனும் மதிப்பின் கணிப்பும் 1 என அமைந்தால் = True Positive

0 எனும் மதிப்பின் கணிப்பும் 0 என அமைந்தால் = True Negative





Precision, Recall & F1 score

Precision (P) என்பது எத்தனை சதவீதம் தவறாக 'ஆம்' எனக் கணித்துள்ளது என்பதையும்,

Recall (R) என்பது எத்தனை சதவீதம் தவறாக 'இல்லை' எனக் கணித்துள்ளது என்பதையும் கணக்கிடுகிறது. தவறாக கணிக்கப்பட்ட இவ்விரண்டு மதிப்புகளையும் சேர்த்து ஒரே மதிப்பாக மாற்றுவதே F score ஆகும். இதற்கான சூத்திரம் பின்வருமாறு.

P = True Positive / (True Positive+ False Positive)

R = True Positive / (True Positive+ False Negative)

F score = 2 (PR / P+R)

இவைகளைக் கண்டு பிடிப்பதற்கான முக்கியத்துவம் என்ன என்று இப்போது பார்க்கலாம். உதாரணத்துக்கு ஒருவருக்கு உடம்பில் ஏற்பட்டுள்ள கட்டியின் அளவைப் பொருத்து, அது புற்று

நோய்க்கான கட்டியா இல்லையா என முடிவு செய்யும் சோதனையை எடுத்துக் கொள்வோம். இதற்கான மாதிரித் தரவுகளில் நூற்றில் ஒருவருக்கு மட்டுமே 'ஆம்' எனும் முடிவு காணப்படும். பெரும்பான்மையான தரவுகளில் 'இல்லை' எனும் முடிவே நிறைந்திருக்கும். இது போன்று ஒரே முடிவினைச் சார்ந்த அதிக அளவு மாதிரித் தரவுகளைக் கொண்டவையே *"skewed classes"* என்றழைக்கப்படுகின்றன. இவற்றை வைத்து பிற்காலத்தில் உண்மையான கட்டியின் அளவைக் கணிக்கும்போது, 'ஆம்' என்பதற்கு பதிலாக 'இல்லை' என்பதையே பெரும்பான்மையாக வெளிப்படுத்தும். இவற்றைக் கண்டறிவதற்கு உதவுவதே *precision* மற்றும் *recall* ஆகும்.

Trading-off between Precision & Recall

ஒருவருடைய கட்டியின் அளவு *5mm* -க்கு மேல் இருந்தால் அது புற்று நோய்க்கான கட்டி என *threshold* அமைக்கப்பட்டுள்ளதாக வைத்துக் கொள்வோம். இப்போது இந்த அளவுக்கு மேல் ஆனால் சாதாரண கட்டி இருக்கும் ஒருவரிடம் சென்று 'இது புற்று நோய்க்கான கட்டி' எனத் தவறாகக் கூறி விட்டால், அவர் தேவையில்லாமல் பல வலிமிகு சிகிச்சைகளை மேற்கொள்ள வேண்டியிருக்கும் *(false positive – high precision).*

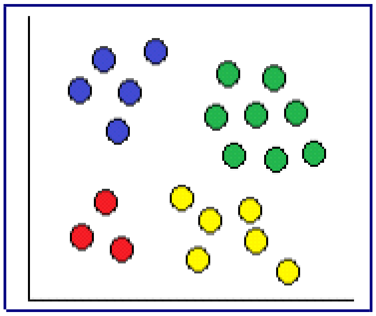
எனவே நமக்கு உறுதியாகத் தெரிந்தால் மட்டுமே 'ஆம்' எனக் கூற வேண்டும் என்பதற்காக *threshold*-ஐ *7mm* -க்கு மேல் அதிகப்படுத்துவோம். இப்போது *6mm* அளவில் புற்று நோய் கட்டி இருக்கும் ஒருவரிடம் சென்று உங்களுக்கு ஒன்றும் 'இல்லை' எனக் கூறும் அபாயம் நேரும் *(false negative – high recall).* இதனால் அவரும் அலட்சியமாக இருந்து விடுவார்.

ஆகவே *precision* -ஐக் குறைக்க விரும்பினால், *recall* அதிகரிக்கும். *Recall*-ஐக் குறைக்க விரும்பினால் *precision* அதிகரிக்கும். இதுவே *trading-off between precision & recall* எனப்படுகிறது.

Multi-class classification

0 மற்றும் *1* என இரு பிரிவுகள் மட்டும் இல்லாமல், பல்வேறு பிரிவுகள் இருப்பின், புதிதாக வரும் ஒன்றினை எந்த பிரிவின் கீழ் அமைக்க வேண்டும் என கணிப்பதே *multi-class classification* ஆகும். இதில் எத்தனை பிரிவுகள் இருக்கிறதோ, அத்தனை *logistic* கணிப்புகள் நடைபெறும். பின்னர் புதிதாக வருகின்ற ஒன்று, அனைத்தினாலும் கணிக்கப்பட்டு , எதில் அதிகமாகப் பொருந்துகிறதோ, அந்தப் பிரிவைச் சென்றடையும்.

கீழ்க்கண்ட உதாரணத்தில் சிகப்பு, ஊதா, பச்சை, மஞ்சள் எனும் நான்கு பிரிவுகளில் வளையங்கள் உள்ளன.

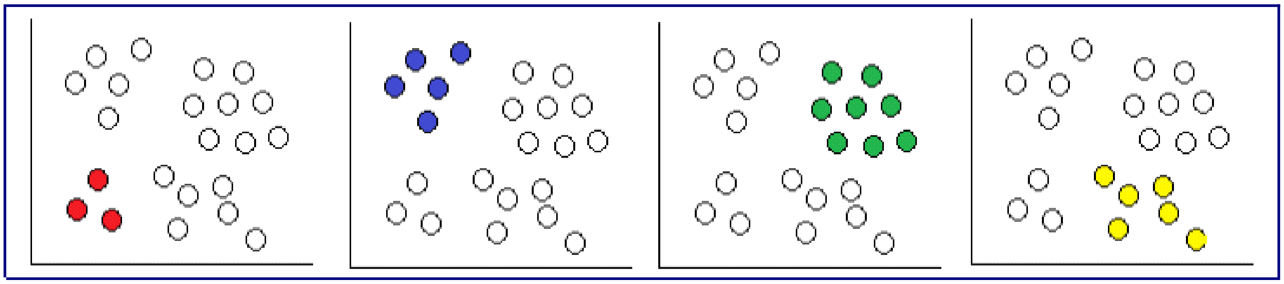


முதலில் சிகப்பினைக் கணிப்பதற்கான *hypothesis* உருவாக்கப்படும். இதில் $h(x) = 1$ என்பது சிகப்பினைக் குறிக்கும். சிகப்பு அல்லாத அனைத்தும் *0* -ஆல் குறிக்கப்படும்.

அடுத்து ஊதாவைக் கணிப்பதற்கான *hypothesis* உருவாக்கப்படும். இதில் $h(x) = 1$ என்பது ஊதாவைக்

குறிக்கும். ஊதா அல்லாத அனைத்தும் 0 -ஆல் குறிக்கப்படும்.

இவ்வாறாக அடுத்தடுத்த நிறங்களுக்கு hypothesis உருவாக்கப்படும்.



பின்னர், புதிதாக ஒரு வளையம் வருகிறதெனில் அது சிகப்பாக கணிக்கப்படுவதற்கான சாத்தியம் 30%, ஊதாவாக கணிக்கப்படுவதற்கான சாத்தியம் 40%, பச்சையாக கணிக்கப்படுவதற்கான சாத்தியம் 60% மஞ்சளாக கணிக்கப்படுவதற்கான சாத்தியம் 50% என வருகிறததேனில் தெ , எதன் சாத்தியம் அதிகமாக இருக்கிறதோ, அந்தப் பிரிவின் கீழ் அமையும். இதுவே multi-class classification ஆகும்.

Decision tree, gaussian NB, KNN, SVC ஆகியவை இதுபோன்ற multi class -க்கு துணைபுரியும் algorithmns ஆகும். ஒரு மலர் மல்லியா, ரோஜாவா, தாமரையா என்று தீர்மானிப்பதற்கான multi-class classification பின்வருமாறு. இவை பல்வேறு algorithms மூலம் நிகழ்த்தப்படுகின்றன. இவைகளில் அதிகமான score மற்றும் precision&recall கொண்டதை நாம் தேர்வு செய்யலாம்..

https://gist.github.com/nithyadurai87/aaded978eb7e545006ed6117c97b86b3

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_recall_fscore_support
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB

df = pd.read_csv('./flowers.csv')
X = df[list(df.columns)[:-1]]
y = df['Flower']
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 0)

tree = DecisionTreeClassifier(max_depth = 2).fit(X_train, y_train)
tree_predictions = tree.predict(X_test)
print (tree.score(X_test, y_test))
print (confusion_matrix(y_test, tree_predictions))
print (precision_recall_fscore_support(y_test, tree_predictions))

svc = SVC(kernel = 'linear', C = 1).fit(X_train, y_train)
svc_predictions = svc.predict(X_test)
print (svc.score(X_test, y_test))
print (confusion_matrix(y_test, svc_predictions))
print (precision_recall_fscore_support(y_test, svc_predictions))

knn = KNeighborsClassifier(n_neighbors = 7).fit(X_train, y_train)
knn_predictions = knn.predict(X_test)
print (knn.score(X_test, y_test))
print (confusion_matrix(y_test, knn_predictions))
print (precision_recall_fscore_support(y_test, knn_predictions))
```

```
gnb = GaussianNB().fit(X_train, y_train)
gnb_predictions = gnb.predict(X_test)
print (gnb.score(X_test, y_test))
print (confusion_matrix(y_test, gnb_predictions))
print (precision_recall_fscore_support(y_test, gnb_predictions))
```

வெளியீடு:

0.8947368421052632
[[15 1 0]
[3 6 0]
[0 0 13]]
(array([0.83333333, 0.85714286, 1.]), array([0.9375, 0.66666667, 1.]), array([0.88235294, 0.75, 1.]), array([16, 9, 13], dtype=int64))

0.9736842105263158
[[15 1 0]
[0 9 0]
[0 0 13]]
(array([1. , 0.9, 1.]), array([0.9375, 1. , 1.]), array([0.96774194, 0.94736842, 1.]), array([16, 9, 13], dtype=int64))

0.9736842105263158
[[15 1 0]
[0 9 0]
[0 0 13]]
(array([1. , 0.9, 1.]), array([0.9375, 1. , 1.]), array([0.96774194, 0.94736842, 1.]), array([16, 9, 13], dtype=int64))

1.0
[[16 0 0]
[0 9 0]
[0 0 13]]
(array([1., 1., 1.]), array([1., 1., 1.]), array([1., 1., 1.]), array([16, 9, 13], dtype=int64))

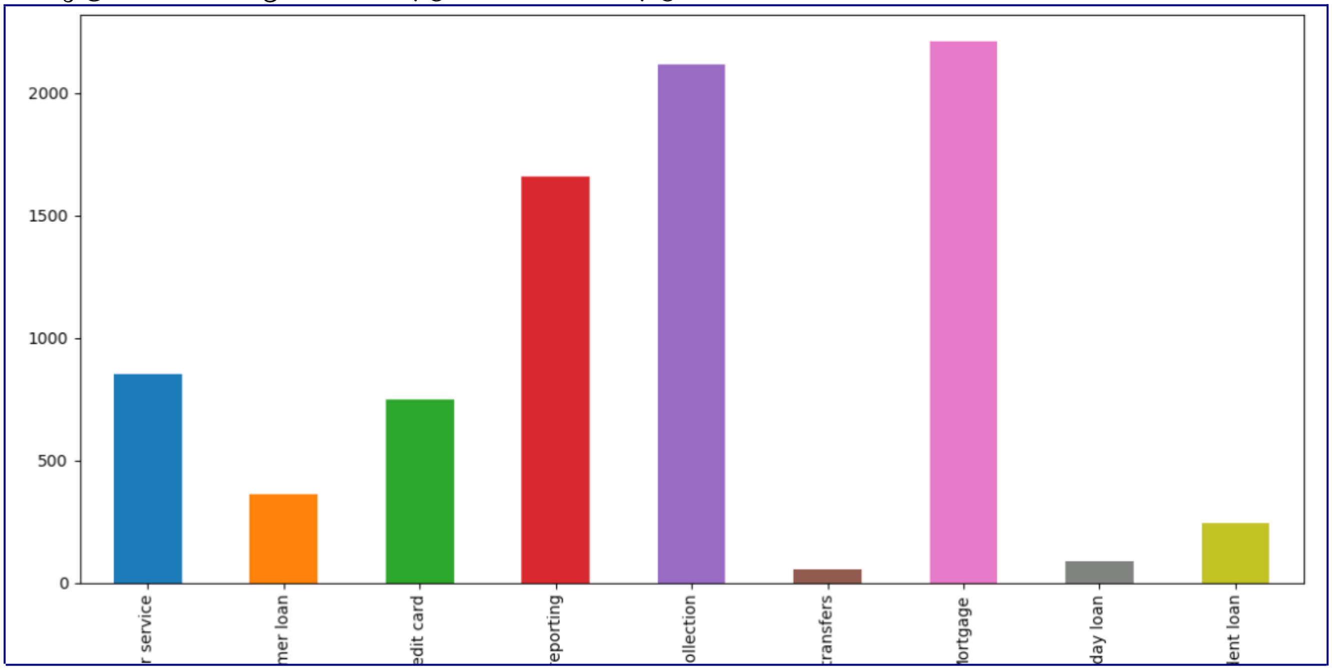
அடுத்ததாக வாடிக்கையாளர் புகாரில் உள்ள வார்த்தைகளைக் கொண்டு, அந்தப் புகார் எந்த வகையின் கீழ் அமையும் என கணிக்கும் *MultinomialNB algorithm* பின்வருமாறு.

https://gist.github.com/nithyadurai87/3ce9dab55025fe1fd41b4da48d3fcbd8

```
import pandas as pd
from io import StringIO
import matplotlib.pyplot as plt
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_selection import chi2
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.naive_bayes import MultinomialNB

df = pd.read_csv('./Consumer_Complaints.csv', sep=',', error_bad_lines=False,
index_col=False, dtype='unicode')
df = df[pd.notnull(df['Issue'])]

fig = plt.figure(figsize=(8,6))
```

```
df.groupby('Product').Issue.count().plot.bar(ylim=0)
plt.show()

X_train, X_test, y_train, y_test = train_test_split(df['Issue'], df['Product'],
random_state = 0)
c = CountVectorizer()
clf = MultinomialNB().fit
(TfidfTransformer().fit_transform(c.fit_transform(X_train)), y_train)

print(clf.predict(c.transform(["This company refuses to provide me verification
and validation of debt per my right under the FDCPA. I do not believe this debt
is mine."])))

tfidf = TfidfVectorizer(sublinear_tf=True, min_df=5, norm='l2', encoding='latin-
1', ngram_range=(1, 2), stop_words='english')
features = tfidf.fit_transform(df.Issue).toarray()
print (features)
df['category_id'] = df['Product'].factorize()[0]
pro_cat = df[['Product',
'category_id']].drop_duplicates().sort_values('category_id')
print (pro_cat)
for i, j in sorted(dict(pro_cat.values).items()):
      indices = np.argsort(chi2(features, df.category_id == j)[0])
      print (indices)
      feature_names = np.array(tfidf.get_feature_names())[indices]
      unigrams = [i for i in feature_names if len(i.split(' ')) == 1]
      bigrams = [i for i in feature_names if len(i.split(' ')) == 2]
      print(">",i)
      print("unigrams:",','.join(unigrams[:5]))
    print("bigrams:",','.join(bigrams[:5]))
```

இதற்கு முதலில் ஒவ்வொரு product -ன் கீழும் எத்தனை புகார்கள் பயிற்சிக்குக் கொடுக்கப்பட்டுள்ளன என ஒரு வரைபடம் மூலம் வரைந்து பார்க்கப்படுகிறது.



பின்னர் அவை 70-30 எனும் விகிதத்தின் படி பயிற்சி கொடுக்கப்பட்டு சோதிக்கப்படுகிறது.

இதில் *TfidfVectorizer* மூலம் புகாரில் உள்ள தனித்தனி வார்த்தைகள் அனைத்தும் *features* -ஆக சேமிக்கப்படுகின்றன. பின்னர் *chi2* மூலம் ஒவ்வொரு தனித்தனி *category* -யோடும் தொடர்பு கொண்டுள்ள வார்த்தைகளின் பட்டியல் சேமிக்கப்படுகிறது. பின்னர் அவை தனித்தனி வார்த்தையாக அமைந்தால் எந்த *category* -ன் கீழ் அமையும், இரண்டிரண்டாக அமைந்தால் எந்த *category* -ன் கீழ் அமையும் என்பது *unigrams, bigrams* எனும் பெயரில் சேமிக்கப்படுகின்றன..

Vectors

classification problem என்பது 'ஆம்' அல்லது 'இல்லை' எனும் மதிப்பின் கீழ் கணிப்பினை நிகழ்த்தும் என ஏற்கனவே கண்டோம். இவை முறையே *1* அல்லது *0*-ஆல் குறிக்கப்படும். நாம் சிலசமயம் வாக்கியங்களையோ, நிழற்படங்களையோ, ஓவியங்களையோ உள்ளீடாகக் கொடுத்து பயிற்சி அளிக்க வேண்டியிருக்கும். இதுபோன்ற இடங்களில் இவற்றையெல்லாம் *1's & 0's* -ஆக மாற்றுவதற்கு *sklearn* வழங்குகின்ற பல்வேறு வகையான வெக்டர்கள் பற்றியும் அவற்றின் பயன்பாடுகள் பற்றியும் பின்வருமாறு காணலாம்.

பல்வேறு வாக்கியங்களைப் பெற்றிருக்கும் ஒரு தொகுப்பு *corpus* எனப்படுகிறது. இந்த *corpus*-ல் உள்ள அனைத்தையும் *0's & 1's* ஆக மாற்றுவதற்கு *dictvectorizer() , countvectorizer()* ஆகியவை பயன்படுகின்றன.

கீழ்க்கண்ட உதாரணத்தில் *corpus1* மற்றும் *corpus2* எனும் இரண்டு *corpus* கொடுக்கப்பட்டுள்ளன. முதலில் உள்ளது *dictvectorizer()* -க்கு உதாரணமாகவும், இரண்டாவதாக உள்ளது *countvectorizer()* -க்கு உதாரணமாகவும் அமைந்துள்ளது. அடுத்ததாக *vector* எனும் *variable*-ல், *corpus2*-ல் உள்ள வாக்கியங்களுக்கான *encode* செய்யப்பட்ட வெக்டர்கள் அமைந்துள்ளன. இவற்றை வைத்து நாம் இரண்டு வெக்டர்களுக்கிடையேயான *euclidean distance*-ஐ எவ்வாறு கண்டுபிடிப்பது என்று பார்க்கலாம்.

https://gist.github.com/nithyadurai87/f3fff58ab7272279ef069689fc391dec

```
from sklearn.feature_extraction import DictVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import HashingVectorizer
from sklearn.metrics.pairwise import euclidean_distances

corpus1 = [{'Gender': 'Male'},{'Gender': 'Female'},{'Gender': 'Transgender'},
{'Gender': 'Male'},{'Gender': 'Female'}]
corpus2 = ['Bird is a Peacock Bird','Peacock dances very well','It eats variety
of seeds','Cumin seed was eaten by it once']
vectors = [[2, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0],[0, 0, 0, 1, 0, 0,
0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1],
 [0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0],[0, 1, 1, 0, 1, 0, 0, 1, 0,
1, 0, 1, 0, 0, 0, 1, 0]]

# one-hot encoding
v1 = DictVectorizer()
print (v1.fit_transform(corpus1).toarray())
print (v1.vocabulary_)

# bag-of-words (term frequencies, binary frequencies)
v2 = CountVectorizer()
print (v2.fit_transform(corpus2).todense())
print (v2.vocabulary_)

print (TfidfVectorizer().fit_transform(corpus2).todense())

print (HashingVectorizer(n_features=6).transform(corpus2).todense())
```

```
print (euclidean_distances([vectors[0]],[vectors[1]]))
print (euclidean_distances([vectors[0]],[vectors[2]]))
print (euclidean_distances([vectors[0]],[vectors[3]]))
```

1. dictvectorizer() -ஒரு *categorical variable*-ஐ *1's & 0's* -ஆக மாற்ற உதவும். இங்கு *'Gender'* எனும் *categorical variable*-ன் மதிப்பாக *'Male', 'Female', 'Transgender'* ஆகியவை அமைந்துள்ளன. முதலில் இத்தகைய *unique* மதிப்புகளை வைத்து ஒரு *dictionary*-ஐ உருவாக்கும். பின்னர் இந்த *3* தனித்தனி வார்த்தைகளும், அவற்றைப் பெற்று விளங்கும் *5* வரிகளும் *5*3 dimension* கொண்ட ஒரு *matrix*-ஆக உருவாக்கப்படும். அதாவது ஒவ்வொரு வரியும் அந்த *matrix*-ன் ஒரு *row* ஆகவும், அந்த வரியில் *dictionary*-ல் உள்ள வார்த்தை இடம்பெற்றிருப்பின் *1* எனவும், இல்லையெனில் *0* எனவும் போட்டு வைத்துக்கொள்ளும். இவ்வாறே ஒரு வெக்டர் உருவாக்கப்படுகிறது. இதுவே *one-hot encoding* எனப்படுகிறது.

```
print (v1.fit_transform(corpus1).toarray())
[[0. 1. 0.]
 [1. 0. 0.]
 [0. 0. 1.]
 [0. 1. 0.]
 [1. 0. 0.]]
```

fit_transform() என்பது நமது *corpus*-ஐ உள்ளீடாக எடுத்துக்கொண்டு வெக்டருக்குக் கற்றுக்கொடுக்கும். *to_dense()* என்பது வார்த்தைகளின் அடர்த்திக்கான வெக்டரை உருவாக்கும். *_vocabulary* என்பது நமது வெக்டர் உருவாக்கத்திற்கு உதவிய *dictionary*-ஐக் கொண்டிருக்கும்.

```
print (v1.vocabulary_)
{'Gender=Male': 1, 'Gender=Female': 0, 'Gender=Transgender': 2}
```

2. countvectorizer() - கொடுக்கப்பட்ட வாக்கியங்கள் அனைத்தையும் *1's & 0's* -ஆக மாற்றும். நமது உதாரணத்தில் *4* வரிகளும், *17* தனித்துவ வார்த்தைகளும் உள்ளன. எனவே *4*17 dimension* கொண்ட *matrix* உருவாக்கப்பட்டுள்ளது. ன் ஒவ்வொரு வரியிலும் எந்தெந்த வார்த்தை இடம்பெற்றுள்ளதோ அது *1* எனவும், இடம்பெறாத வார்த்தை *0* எனவும் அமைந்திருப்பதைக் காணலாம். இதுவே *bag of words* எனப்படுகிறது.

வார்த்தைகள் அமைந்திருக்கும் அதே வரிசையில்தான் *encode* செய்யப்பட்டவை இடம் பெற்றிருக்கும் எனக் கூற முடியாது. வார்த்தைகளில் உள்ள எல்லா எழுத்துக்களையும், சிறிய எழுத்துக்களாக மாற்றிவிட்டு அதனை *tokens*-ஆக மாற்றும். *Tokenization* என்பது இரண்டுக்கும் மேற்பட்ட எழுத்துக்களைப் பெற்றிருக்கும் வார்த்தைகளை இடைவெளி வைத்துப் பிரித்து *tokens*-ஆக மாற்றுவதே ஆகும். *Tokens* என்பது கோப்பினுள் இடம் பெற்றுள்ள வார்த்தைகள் ஆகும்.

Bird is a Peacock Bird','Peacock dances very well','It eats variety of seeds','Cumin seed was eaten by it once'

```
print (v2.fit_transform(corpus2).todense())
[[2 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0]
```

```
[0 0 0 1 0 0 0 0 0 0 1 0 0 0 1 0 1]
[0 0 0 0 0 1 0 1 1 0 0 0 1 1 0 0 0]
[0 1 1 0 1 0 0 1 0 1 0 1 0 0 0 1 0]]
```

இதனை *binary frequency* மற்றும் *term frequency* என்னும் இரண்டு விதங்களில் குறிப்பிடலாம். *binary* என்பது வெறும் *1's & 0's* -ஐ மட்டும் வெளிப்படுத்தும். *term* என்பது ஒவ்வொரு வார்த்தையும் எத்தனை முறை இடம்பெற்றுள்ளது என்பதை வெளிப்படுத்தும். இங்கு *Bird* என்பது முதல் வாக்கியத்தில் இரண்டு முறை உள்ளதால் அந்த இடத்தில் *2* என வெளிப்படுத்தப்பட்டுள்ளதைக் காணவும்.

இதற்கான *vocabulary*-ல் அந்த வரியிலிருந்தும் எடுக்கப்பட்ட *17* தனித்துவ வார்த்தைகள் அமைந்திருப்பதைக் காணவும் *(0* முதல் *16* வரை*)*. இங்கு *Bird, Peacock, it* ஆகிய வார்த்தைகள் இரண்டு முறை இடம்பெற்றுள்ளது. ஆனால் ஒரே ஒரு முறை மட்டும் தான் இங்கு சேமிக்கப்பட்டுள்ளது. அவ்வாறே *case-sensitive* இல்லாமல் *it, It* ஆகிய இரண்டும் ஒன்றாக எடுத்துக்கொள்ளப்பட்டுள்ளது. மேலும் *a* என்பது ஒரு தனி வார்த்தையாக எடுத்துக்கொள்ளப்படவில்லை.

```
print (v2.vocabulary_)
{'bird': 0, 'is': 6, 'peacock': 10, 'dances': 3, 'very': 14, 'well': 16, 'it': 7
, 'eats': 5, 'variety': 13, 'of': 8, 'seeds': 12, 'cumin': 2, 'seed': 11, 'was':
 15, 'eaten': 4, 'by': 1, 'once': 9}
```

3. TfidfVectorizer() - Term frequency மூலம் உருவாக்கப்படும் வெக்டரை *normalize* செய்து அந்த *frequency* -க்கான *weight*-ஐ வெளிப்படுத்தும். வெறும் *raw count*-ஆக *2* என வெளிப்படுத்தாமல், அதனை *normalize* செய்து வெளிப்படுத்துவதே *L2 Normalization (level2)* எனப்படும்.

```
print (TfidfVectorizer().fit_transform(corpus2).todense())
[[0.84352956 0.         0.         0.         0.         0.
  0.42176478 0.         0.         0.         0.3325242  0.
  0.         0.         0.         0.         0.        ]
 [0.         0.         0.         0.52547275 0.         0.
  0.         0.         0.         0.         0.41428875 0.
  0.         0.         0.52547275 0.         0.52547275]
 [0.         0.         0.         0.         0.         0.46516193
  0.         0.36673901 0.46516193 0.         0.         0.
  0.46516193 0.46516193 0.         0.         0.        ]
 [0.         0.38861429 0.38861429 0.         0.38861429 0.
  0.         0.30638797 0.         0.38861429 0.         0.38861429
  0.         0.         0.         0.38861429 0.        ]]
```

4. HashingVectorizer() - அகராதியின் துணை இல்லாமலேயே நேரடியாக வெக்டரை உருவாக்கும்.. மேற்கண்ட *dict & count* ஆகிய இரண்டும் *2* படிகளில் வேலை செய்யும். முதலில் வெக்டர் உருவாக்கத்திற்குத் தேவையான *dictionary*-யை உருவாக்கும். அடுத்தபடியாகத்தான் வெக்டரை உருவாக்கும். இதில் முதல் படியைத் தவிர்த்து நேரடியாக வெக்டரை உருவாக்குவதைத்தான் *Hashing Trick* என்போம். ஏனெனில் *dictionary*-ன் அளவு பெருகப் பெருக அந்தளவுக்குப் பெரிய அகராதியை சேமிக்கத் தேவையான *memory*-ன் அளவும் அதிகரிக்கும். இதைத் தவிர்ப்பதற்காக வந்ததே

இவ்வகையான வெக்டர் ஆகும்.

```
print (HashingVectorizer(n_features=6).transform(corpus2).todense())
[[ 0.         -0.70710678 -0.70710678  0.          0.          0.        ]
 [ 0.          0.         -0.81649658 -0.40824829  0.40824829  0.        ]
 [ 0.75592895  0.         -0.37796447  0.         -0.37796447 -0.37796447]
 [ 0.25819889  0.77459667  0.         -0.51639778  0.          0.25819889]]
```

5. euclidean_distances - encode செய்யப்பட்ட இரண்டு வாக்கியங்களுக்கிடையேயான வேறுபாடு எந்த அளவுக்கு உள்ளது என்பதைக் கணக்கிட உதவும். மேற்கண்ட உதாரணத்தில் முதல் இரண்டு வாக்கியங்களுக்கு இடையேயான வேறுபாடு சற்று குறைவாகவும், முதலுக்கும் 3-வது வாக்கியத்துக்குமான வேறுபாடு சற்று அதிகமாகவும், முதலுக்கும் 4-வது வாக்கியத்துக்குமான வேறுபாடு இன்னும் சற்று அதிகமாகவும் இருப்பதைக் காணலாம்.

```
print (euclidean_distances([vectors[0]],[vectors[1]]))
[[2.82842712]]

print (euclidean_distances([vectors[0]],[vectors[2]]))
[[3.31662479]]

print (euclidean_distances([vectors[0]],[vectors[3]]))
[[3.60555128]]
```

Natural Language Toolkit

இதுவரை நாம் கண்ட வெக்டர் உருவாக்கம் அனைத்திலும் ஏதேனும் ஓரிரண்டு வார்த்தைகள் மட்டுமே இடம்பெற்றிருந்தாலும் கூட, இடம் பெறாத வார்த்தைகளுக்கான 0's ஐ அது கொண்டிருக்கும். இதனால் அந்த வெக்டருடைய அளவு அதிகரிக்கிறது. இதுபோன்ற அதிக அளவிலான 0's -ஐப் பெற்று விளங்கும் வெக்டர்தான் sparse vector என்று அழைக்கப்படுகிறது. உதாரணத்துக்கு ஒரு கோப்பினுள் அரசியல், சினிமா, விளையாட்டு போன்ற பல்வேறு துறைகளுக்கான வாக்கியங்கள் உள்ளதெனில், அவற்றையெல்லாம் ஒரு வெக்டராக மாற்றும் போது அரசியலுக்கான வரியில் சினிமாவுக்கான வார்த்தை இடம்பெற்றிருக்காது, அதேபோல் சினிமாவுக்கான வரியில் விளையாட்டுக்கான வார்த்தை இடம்பெற்றிருக்காது. இதேபோல் பார்த்தால் ஒவ்வொரு வரியிலும் தேவையில்லாத பல 0's நிறைந்திருக்கும். இதனால் 2 முக்கியப் பிரச்சனைகள் எழுகின்றன.

முதலாவதாக அதிக அளவு memory & space வீணாகிறது. Numpy என்பது 0's அல்லாதவற்றை மட்டும் குறிப்பிடுவதற்காக ஒருசில சிறப்பு வகை தரவு வகைகளை வழங்குகின்றன. அடுத்ததாக dimensionality-ன் அளவு அதிகரிக்க அதிகரிக்க அந்த அளவுக்குப் பயிற்சி அளிக்கத் தேவையான தரவுகளின் எண்ணிக்கையும் அதிகரிக்கிறது. இல்லையெனில் overfit ஆவதற்கான அபாயம் உள்ளது. இதுவே 'curse of dimensionality' அல்லது 'Hughes effect' என்றழைக்கப்படுகிறது. இதை எவ்வாறு குறைப்பது எனப் பேசுவதே dimensionality reduction ஆகும்.

நமது வெக்டர் உருவாக்கத்தின்போது stop_words='english' எனக் கொடுத்தோமானால் is,was,are போன்ற ஆங்கிலத்தில் வருகின்ற துணைச் சொற்களை எல்லாம் தவிர்த்து மீதமுள்ள சொற்களுக்கு மட்டும்

dictionary உருவாக்கப்படும். இதனால் அதன் *dimensionality* குறைகிறது.

அவ்வாறே *NLTK* எனும் கருவி ஒன்று உள்ளது. அதிலுள்ள *stemmer, lemmatizer* ஆகியவற்றைப் பயன்படுத்துவதன் மூலம் வெக்டரின் *dimensionality* இன்னும் குறைக்கப்படுவதைக் காணலாம்.

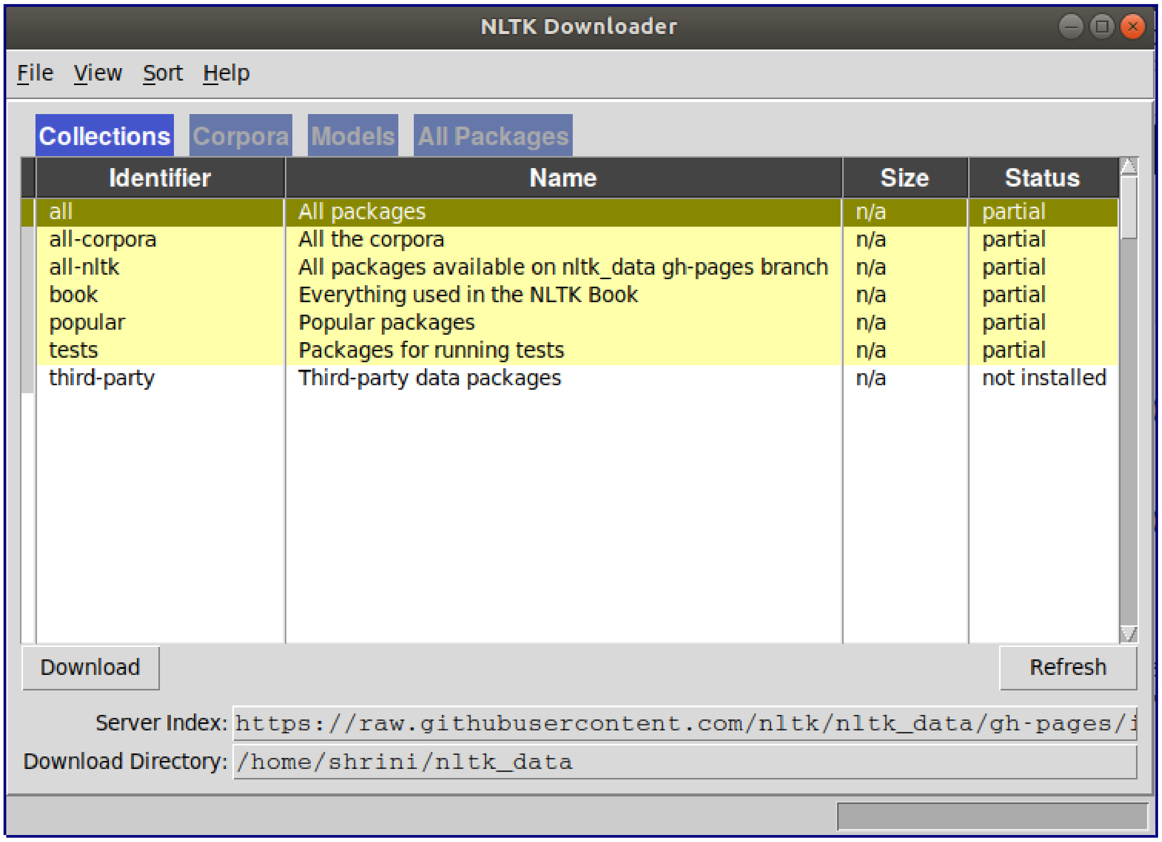
[*https://gist.github.com/nithyadurai87/491e5e6f9c009ebd88912e71ef9363a4*](https://gist.github.com/nithyadurai87/491e5e6f9c009ebd88912e71ef9363a4)

```
"""
import nltk
nltk.download()
"""

from sklearn.feature_extraction.text import CountVectorizer
from nltk import word_tokenize
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer
from nltk import pos_tag

def lemmatize(token, tag):
	if tag[0].lower() in ['n', 'v']:
		return WordNetLemmatizer().lemmatize(token, tag[0].lower())
	return token

corpus = ['Bird is a Peacock Bird','Peacock dances very well','It eats variety of seeds','Cumin seed was eaten by it once']

print (CountVectorizer().fit_transform(corpus).todense())
print (CountVectorizer(stop_words='english').fit_transform(corpus).todense())

print (PorterStemmer().stem('seeds'))

print (WordNetLemmatizer().lemmatize('gathering', 'v'))
print (WordNetLemmatizer().lemmatize('gathering', 'n'))

s_lines=[]
for document in corpus:
	s_words=[]
	for token in word_tokenize(document):
		s_words.append(PorterStemmer().stem(token))
	s_lines.append(s_words)
print ('Stemmed:',s_lines)

tagged_corpus=[]
for document in corpus:
	tagged_corpus.append(pos_tag(word_tokenize(document)))

l_lines=[]
for document in tagged_corpus:
	l_words=[]
	for token, tag in document:
		l_words.append(lemmatize(token, tag))
	l_lines.append(l_words)
print ('Lemmatized:',l_lines)
```

இதனை பின்வருமாறு பதிவிறக்கம் செய்து பயன்படுத்தலாம்.

```
```

```
import nltk
nltk.download()
```



'Bird is a Peacock Bird','Peacock dances very well','It eats variety of seeds',
'Cumin seed was eaten by it once'

*1. மேற்கண்ட வாக்கியங்களுக்கான CountVectorizer() என்பது பின்வருமாறு ஒரு வெக்டரை உருவாக்கும்(4*17).*

```
print (CountVectorizer().fit_transform(corpus).todense())

[[2 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0]
 [0 0 0 1 0 0 0 0 0 0 1 0 0 0 1 0 1]
 [0 0 0 0 0 1 0 1 1 0 0 0 1 1 0 0 0]
 [0 1 1 0 1 0 0 1 0 1 0 1 0 0 0 1 0]]
```

மேற்கண்ட அதே வாக்கியங்களுக்கு stop_words='english' எனக் கொடுத்து வெக்டர் உருவாக்கும்போது, is, very, well, it, of, was, by, once ஆகிய வார்த்தைகள் நீக்கப்படுவதால் dimensionality

*குறைந்து இருப்பதைக் காணலாம்(4*9).*

```
print (CountVectorizer(stop_words='english').fit_transform(corpus).todense())
[[2 0 0 0 0 1 0 0 0]
 [0 0 1 0 0 1 0 0 0]
 [0 0 0 0 1 0 0 1 1]
 [0 1 0 1 0 0 1 0 0]]
```

2. stop_words='english' பயன்படுத்தினாலும் கூட, seeds, seed ஆகிய இரண்டும் இரண்டு தனித்தனி வார்த்தைகளாக சேமிக்கப்படுகின்றன. இதைத் தவிர்ப்பதற்காக வந்ததே PorterStemmer() ஆகும். இது ஒரு ஆங்கிலச் சொல்லின் வேர்சொல்லை கண்டறிந்து அதை மட்டும் சேமிக்கும். அதைத் தழுவி வருகின்ற இன்ன பிற சொற்களையெல்லாம் சேமிக்காது.

```
print (PorterStemmer().stem('seeds'))
seed
```

3. WordNetLemmatizer() என்பது ஒரு ஆங்கிலச் சொல்லை அதன் பொருளறிந்து பிரித்து சேமிக்கும். அதாவது ஒரே ஒரு சொல் ஓரிடத்தில் பெயர்ச்சொல்லாகவும் மற்றொரு இடத்தில் வினைசொல்லாகவும் பயன்படுத்தப்பட்டிருப்பின் அவை இரண்டையும் இரண்டு தனித்தனி சொற்களாகச் சேமிக்கும். உதாரணத்துக்கு 'I am gathering foods for birds', 'seeds are stored in the gathering place' என்பதில் gathering, gather என்பது இரண்டு தனித்தனி வார்த்தைகளாக சேமிக்கப்படும்.

```
print (WordNetLemmatizer().lemmatize('gathering', 'v'))
gather

print (WordNetLemmatizer().lemmatize('gathering', 'n'))
gathering
```

4. நம்முடைய corpus-ஐ NLTK கொண்டு அணுகும்போது, அது பின்வருமாறு வெளிப்படுத்தும்.

```
print ('Stemmed:',s_lines)
Stemmed: [['bird', 'is', 'a', 'peacock', 'bird'], ['peacock', 'danc', 'veri', 'w
ell'], ['It', 'eat', 'varieti', 'of', 'seed'], ['cumin', 'seed', 'wa', 'eaten',
'by', 'it', 'onc']]

print ('Lemmatized:',l_lines)
Lemmatized: [['Bird', 'be', 'a', 'Peacock', 'Bird'], ['Peacock', 'dance', 'very',
'well'],
['It', 'eat', 'variety', 'of', 'seed'], ['Cumin', 'seed', 'be', 'eat', 'by',
'it', 'once']]
```

Decision Trees & Random Forest

Regression மற்றும் *Classification* இரண்டிற்கும் உதவக்கூடிய நேர்கோடு முறையில் பிரிக்க இயலாத *non-linear* தரவுகளுக்கான *model*-ஆக *decision trees* மற்றும் *random forest* விளங்குகிறது. *Decision trees* என்பது பொதுவாக மாதிரித் தரவுகளில் உள்ள மதிப்புகளைக் கொண்டு அவற்றை சிறுசிறு பகுதிகளாகப் பிரித்துக் கற்கிறது. கீழ்க்கண்ட எடுத்துக்காட்டில் ஒரு மலர் மல்லியா, ரோஜாவா, தாமரையா என்று தீர்மானிக்க *DecisionTreeClassifier()* மற்றும் *RandomForestClassifier()* பயன்படுத்தப்பட்டுள்ளன. ஒவ்வொரு மலரின் இதழ்களுடைய*(sepal)* நீள அகலமும், அவற்றின் மேற்புற இதழ்களுடைய*(petal)* நீள அகலமுமான *4* அம்சங்களே ஒரு மலர் எந்த மலராக இருக்கும் என்பதைத் தீர்மானிக்கிறது. இந்த அம்சங்களிலுள்ள தரவுகளை பல்வேறு பகுதிகளாகப் பிரித்துக் கற்கும் வேலையை *DecisionTreeClassifier()* செய்கிறது. அவ்வாறு தரவுகளைப் பிரிப்பது என்பது ஒருசில *conditions*-ஐப் பொறுத்து நடக்கிறது. எனவேதான் இவை *Eager learners* என்று அழைக்கப்படுகின்றன. இதற்கு மாற்றாக *KNN* என்பது *lazy learners* ஆகும். *Ensemble learning* எனும் முறையில் *random forest* கற்கிறது. *Ensemble* என்றால் குழுமம் என்று பொருள். அதாவது பல்வேறு *decision trees*-ஐ உருவாக்கி, அவற்றை குழுமமாக வைத்துக் கற்கிறது. குழுமத்தில் உள்ள ஒவ்வொரு *tree*-ம் வெவ்வேறு பயிற்சித் தரவுகளை எடுத்துக் கொண்டு பயிற்சி பெற்றுக் கொள்கிறது. எனவே இதனுடைய *accuracy* இன்னும் அதிகமாக இருக்கும். கீழ்க்கண்ட எடுத்துக்காட்டில் இவைகளுக்கான நிரலைக் காணலாம். *Decision Trees 89% accuracy* -ஐயும், *Random forest 97% accuracy* -ஐயும் வெளிப்படுத்துவதைக் காணலாம். மேலும் ஒவ்வொன்றும் எவ்வாறு தரவுகளைப் பிரித்துக் கற்கிறது என்பது வரைபடமாகவும் காட்டப்பட்டுள்ளது.

https://gist.github.com/nithyadurai87/d21ffb25b7f5a38d90a437e9f169d58e

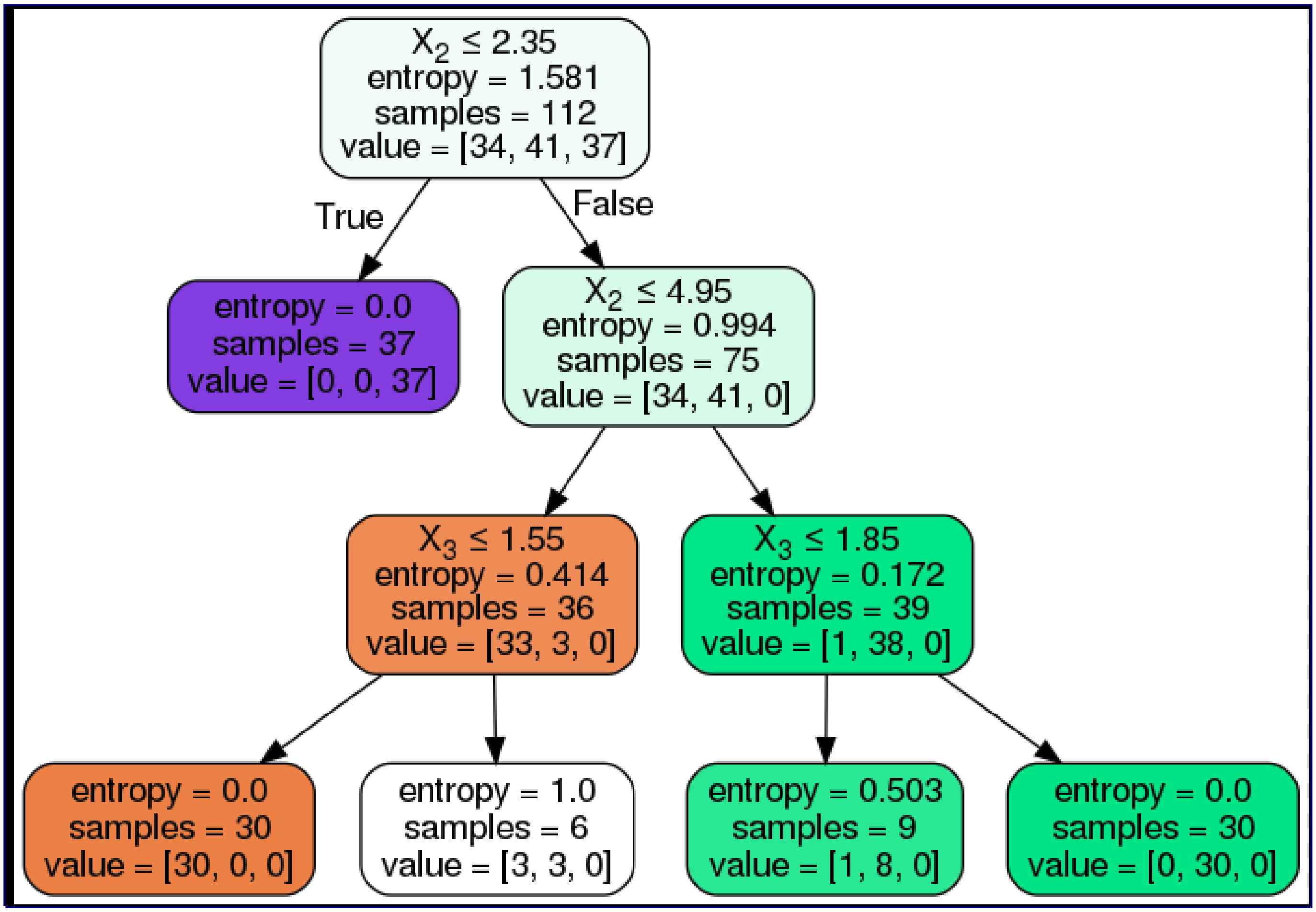
```
from sklearn.datasets import load_iris
import pandas as pd
import os
from sklearn.tree import DecisionTreeClassifier,export_graphviz
from sklearn.metrics import confusion_matrix,accuracy_score,classification_report
from io import StringIO
import pydotplus
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from IPython.display import Image
import matplotlib.pyplot as plt
import seaborn as sns

df = pd.read_csv('./flowers.csv')
X = df[list(df.columns)[:-1]]
y = df['Flower']
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 0)

a = DecisionTreeClassifier(criterion = "entropy", random_state = 100,max_depth=3,
min_samples_leaf=5)  # gini
a.fit(X_train, y_train)
y_pred = a.predict(X_test)
print("Confusion Matrix: ", confusion_matrix(y_test, y_pred))
print ("Accuracy : ", accuracy_score(y_test,y_pred)*100)
print("Report : ", classification_report(y_test, y_pred))

dot_data = StringIO()
export_graphviz(a, out_file=dot_data,filled=True,
rounded=True,special_characters=True)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
graph.write_png("decisiontree.png")

b = RandomForestClassifier(max_depth = None, n_estimators=100)
```

```
b.fit(X_train,y_train)
y_pred = b.predict(X_test)
print("Confusion Matrix: ", confusion_matrix(y_test, y_pred))
print ("Accuracy : ", accuracy_score(y_test,y_pred)*100)
print("Report : ", classification_report(y_test, y_pred))

export_graphviz(b.estimators_[5], out_file='tree.dot', feature_names =
X_train.columns.tolist(),
                class_names = ['Lotus', 'Jasmin', 'Rose'],
                rounded = True, proportion = False, precision = 2, filled = True)

os.system ("dot -Tpng tree.dot -o randomforest.png -Gdpi=600")
Image(filename = 'randomforest.png')
f =
pd.Series(b.feature_importances_,index=X_train.columns.tolist()).sort_values(asce
nding=False)
sns.barplot(x=f, y=f.index)
plt.xlabel('Feature Importance Score')
plt.ylabel('Features')
plt.legend()
plt.show()
```

நிரலுக்கான விளக்கம்:

flowers.csv எனும் கோப்பில் மொத்தம் *150* தரவுகள் பயிற்சிக்கு உள்ளன. அவை *train_test_split()* எனும் முறைப்படி *112* தரவுகள் பயிற்சிக்கும், மீதி *38* தரவுகள் பயிற்சி செய்யப்பட்ட *model*-ஐ சோதிப்பதற்கும் பயன்படுத்தப்பட்டுள்ளன. கீழ்க்கண்ட *decision tree*-ன் முதல் *node*-க்குள் உள்ள *samples=112* என்பது மொத்தம் பயிற்சிக்கு அளிக்கப்பட்டுள்ள தரவுகளைக் குறிக்கிறது. *value = [34,41,37]* என்பது *34* தரவுகள் மல்லிகைக்கும் *41* தரவுகள் தாமரைக்கும், *37* தரவுகள் ரோஜாவுக்கும் அமைந்துள்ளன எனும் விவரத்தைக் கொடுக்கிறது. *entropy = 1.581* மாதிரிகளில் உள்ள *uncertainty / disorder / impurity*-ஐக் குறிக்கிறது. அதாவது நாம் வகைப்படுத்த வேண்டிய பல்வேறு பிரிவுகளில் உள்ள தரவுகளும் எந்த அளவு விகிதத்தில் கலந்துள்ளன என்பதைக் கூறும். இதற்கான கணக்கீடு பின்வரும் முறையில் நிகழும். முதலில் மொத்த தரவுகளில் ஒவ்வொரு பிரிவைச் சேர்ந்த தரவுகளும் எவ்வளவு எண்ணிக்கையில் உள்ளன எனும் பின்னம் கணக்கிடப்படும். பின்னர் அம்மதிப்புக்கு *log base 2* கண்டுபிடிக்க வேண்டும். இதற்கான கருவி *https://www.miniwebtool.com/log-base-2-calculator/* எனும் வலைத்தளத்தில் உள்ளது. இவ்வாறே மல்லி, ரோஜா, தாமரை என்னும் ஒவ்வொரு பிரிவுக்கும் தனித்தனியாகக் கண்டுபிடிக்க வேண்டும். கடைசியாக இவைகளின் கூட்டுத்தொகையை - எனும் எதிர்மறை குறியால் பெருக்கினால் கிடைப்பதே *entropy* ஆகும்.

Entropy = - {Summation of (fraction of each class.log base 2 of that fraction)}
= -{ (34/112).log2(34/112) + (41/112).log2(41/112) + (37/112).log2(37/112) }
= -{ (0.3035).log2(0.3035) + (0.3661).log2(0.3661) + (0.3303).log2(0.3303) }
= -{ (0.3035).(-1.7202) + (0.3661).(-1.4496) + (0.3303).(-1.5981) }
= -{ -0.5220 + -0.5307 + -0.5278 }
= -{ -1.5805}
= 1.581



கணக்கிடப்பட்ட *entropy* மதிப்பையே வரைப்படத்தின் முதல் *node*-ல் காணலாம். இம்மதிப்பு *0*-க்கு அதிகமாக இருப்பதால், *112* தரவுகளும் ஒரு *condition* மூலம் *37, 75* எனும் எண்ணிக்கையில் அமையும் இரு பிரிவுகளாகப் பிரிக்கப்படுகின்றன. அதாவது *X2* எனப்படும் *Petal_length* அம்சத்தின் மதிப்புகளில் *2.35* -க்கு கீழ் இருந்தால் அத்தகைய தரவுகள் இடப்புற *node*-லும், அதிகமாக உள்ளவை வலப்புற *node*-லும் பிரிக்கப்படுகின்றன. பின்னர் மீண்டும் பிரிக்கப்பட்ட இரு பிரிவுகளுக்கும் *entropy* கணக்கிடப்படுகிறது. இடப்புறம் உள்ள *node*-ல் *entropy 0.0* என வந்துள்ளது. இதுவே *decision node* எனப்படும். அதாவது *0*-ஆக இருக்கும் பட்சத்தில் அதில் உள்ள தரவுகள் அனைத்தும் ஏதோ ஒரு வகையின் கீழ் பிரிக்கப்பட்டுவிட்டது என்று அர்த்தம். அதன் *value* மதிப்பும் *[0,0,37]* என்று உள்ளது. அதாவது மல்லிகைக்கும், தாமரைக்குமான தரவுகளின் எண்ணிக்கை *0*. ரோஜாவுக்கான எண்ணிக்கை *37*. இதுவே ஒரு பூவை ரோஜா என முடிவு செய்வதற்கான *decision node* ஆகும். இதே முறையில் வரைப்படத்திலுள்ள மற்ற *nodes* உருவாக்கப்படுகின்றன. மற்ற *features*-ம் சோதிக்கப்படுகின்றன. வரைப்படத்தின் கடைசி கிளையில் ஒரு பூவை மல்லி அல்லது தாமரை என முடிவு செய்வதற்கான *decision nodes* அமைந்துள்ளன. அதாவது கடைசி கிளையில் இடமிருந்து வலமாக உள்ள *3 nodes*-ல், அதன் *value* மதிப்புகளை கவனிக்கவும். மல்லி என முடிவு செய்வதற்கான இடத்தில் *34* என மொத்தமாக இல்லாமல், *30, 3, 1* என தனித்தனியாகப் பிரித்து இத்தகைய *decision nodes*-ஐ உருவாக்கியுள்ளது. அவ்வாறே வலமிருந்து இடமாக உள்ள *3 nodes*-ல், தாமரை என முடிவு செய்வதற்கான இடத்தில் *41* என மொத்தமாக இல்லாமல், *30, 8, 3* எனத் தனித்தனியாகப் பிரித்து உருவாக்கியுள்ளது. எனவேதான் இவைகளின் *entropy 0* மற்றும் அதற்கு நெருங்கிய மதிப்பாக உள்ளது.

Information Gain:

ஒரு குறிப்பிட்ட பிரிவில் தரவுகளை வகைப்படுத்துவதற்குத் தேவையான விவரங்களை எந்த

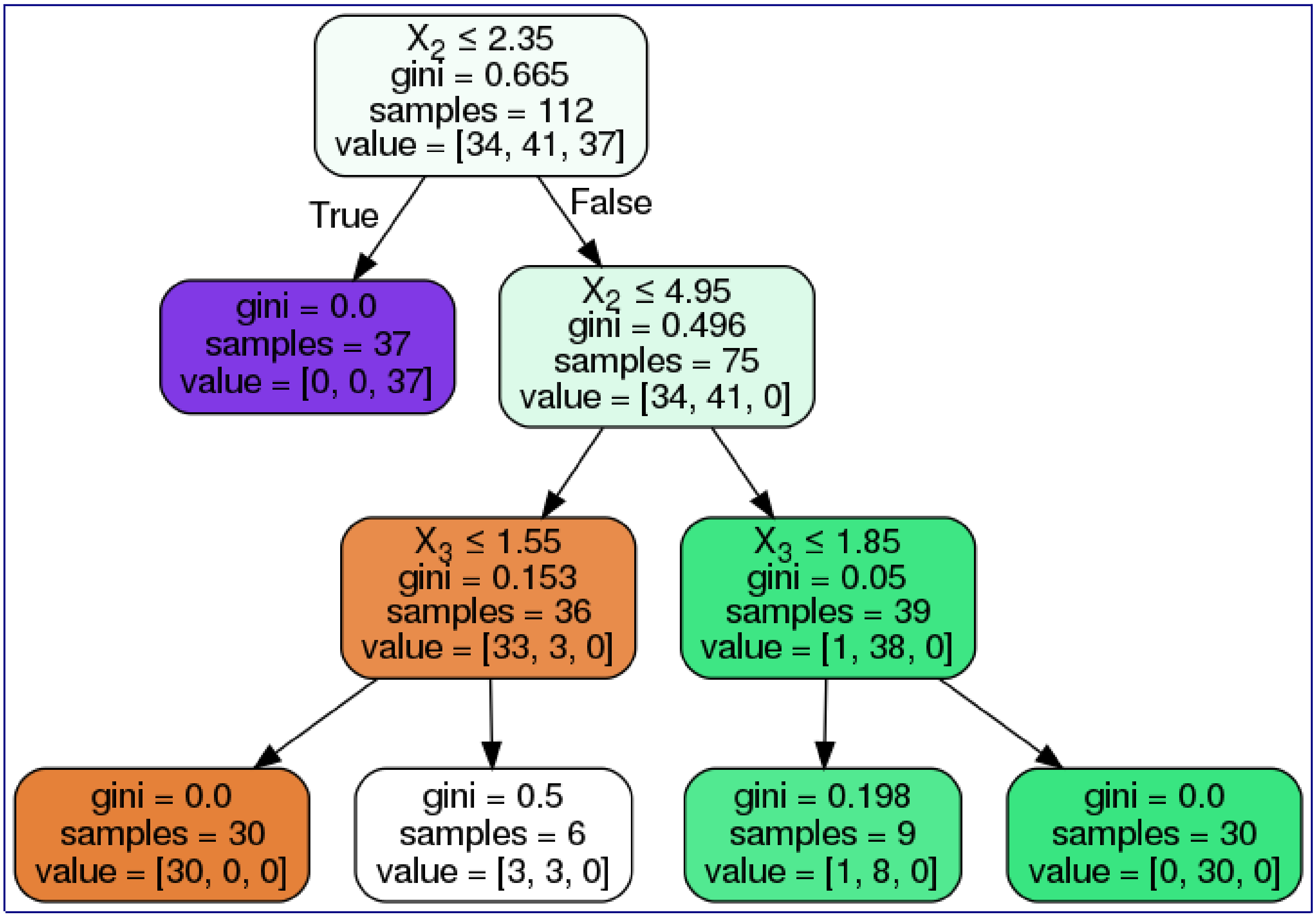
அளவுக்கு ஒரு *feature* அளிக்கிறது என்பதே *Information Gain* எனப்படும். இதுவும் *entropy*-ஐப் போன்றே தரவுகளை சரியாக வகைப்படுத்த உதவும் ஒரு *metric* ஆகும். *entropy* என்பது *impurity* ஆகும். இதை வைத்து, அந்த *impurity*-ஐக் குறைப்பதற்கு உதவும் *metric* தான் *gini gain* எனப்படும். இதற்கான வாய்ப்பாடு பின்வருமாறு.

Information Gain = Parent's entropy - child's entropy with weighted average

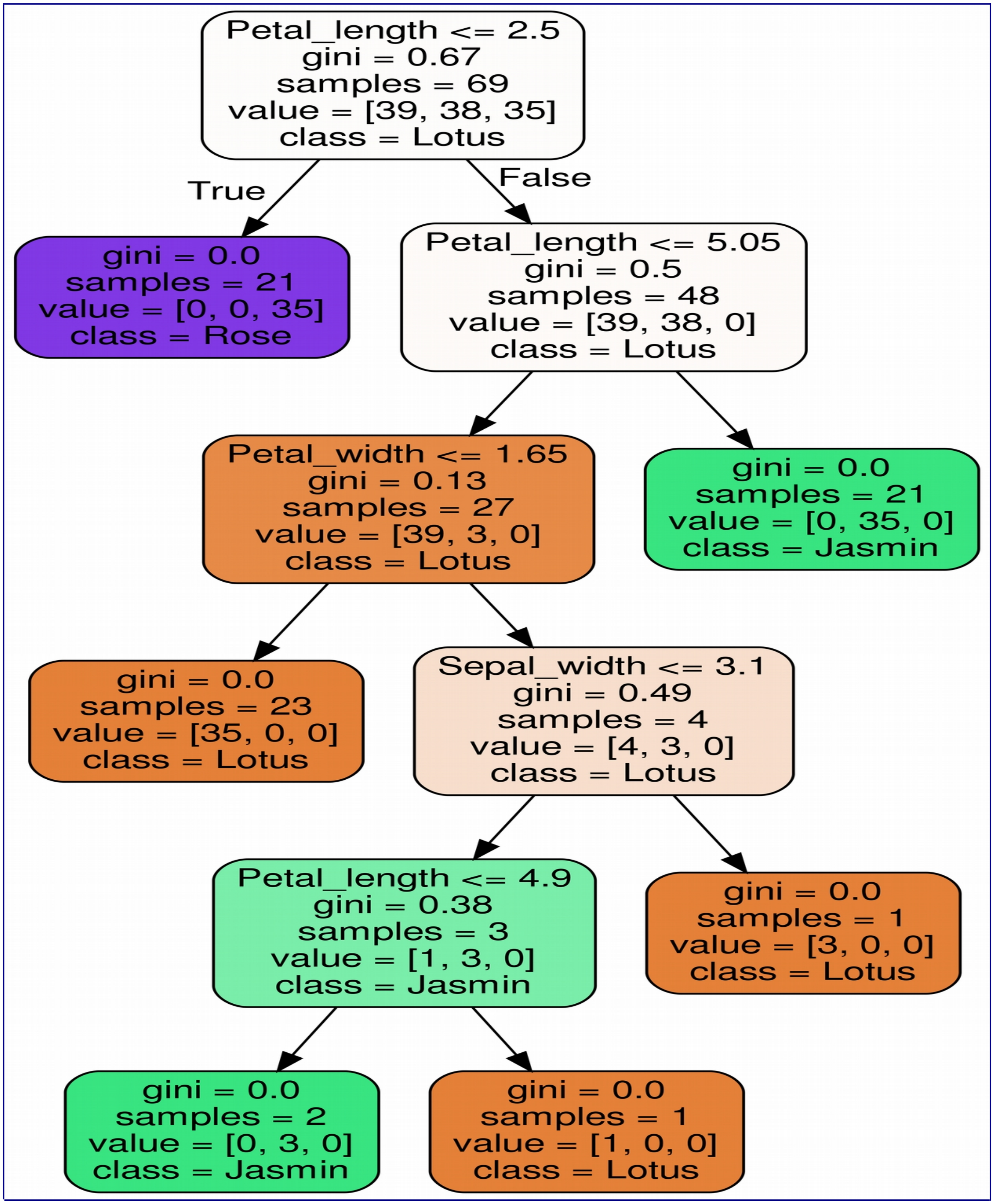
*child's entropy with weighted average = [(no. of examples in left child node) / (total no. of examples in parent node) * (entropy of left node)] +*
*[(no. of examples in right child node)/ (total no. of examples in parent node) * (entropy of right node)]*

*= (37/112)*0.0 + (75/112)*0.994*
= 0 + 0.665625
= 0.665

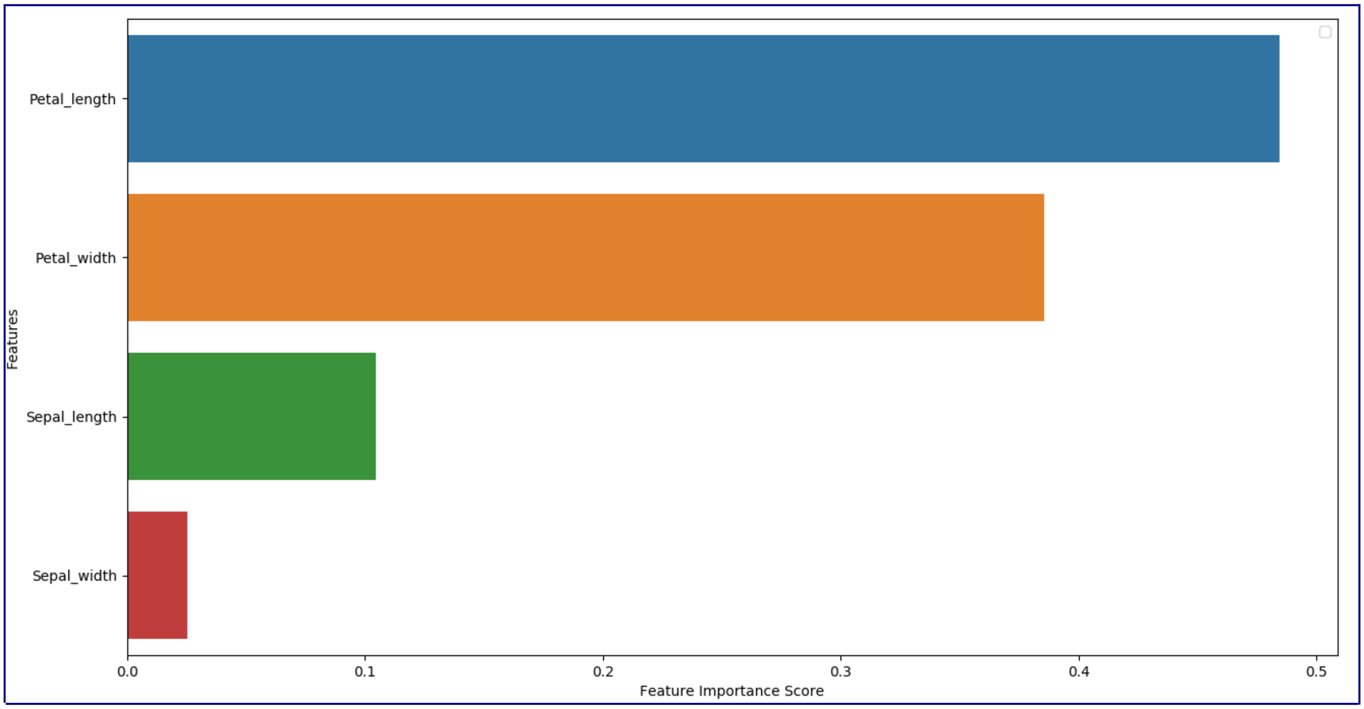
மேற்கண்ட நிரலில் *DecisionTreeClassifier()*-க்குள் *criterion = "entropy"* என்பதற்கு பதிலாக *"gini"*எனக் கொடுத்து பயிற்சி அளித்தால், அது *gini*-ஐக் கணக்கிட்டு பின்வருமாறு கிளைகளை உருவாக்கிக் கற்கிறது.



Random Forest முறையில் கற்கும் *model*-ன் வரைபடம் பின்வருமாறு.



Random forest-ல் மாதிரித் தரவுகளில் உள்ள ஒவ்வொரு *feature*-ம் வகைப்படுத்தலுக்கு எந்த அளவுக்கு பங்களித்துள்ளது என்பதை பின்வரும் வரைப்படத்தில் காணலாம்..



Clustering with K-Means

Unsupervised learning-ல் நாம் கற்க இருக்கும் முதல் *algorithm* இதுவே. இதுவரை நாம் கண்ட அனைத்தும் *supervised*-ன் கீழ் அமையும். *logistic regression, multi-class classification* போன்ற அனைத்திலும், உள்ளீடு(*X*) மற்றும் வெளியீடு(*Y*) இரண்டையும் கொடுத்து பயிற்சி அளிப்போம். பல்வேறு வெளியீட்டு வகைகளின் கீழ் தரவுகளைப் பிரிப்பதற்கு அத்தனை வகையான எல்லைகளையும் நாமே வரையறை செய்வோம். ஆனால் இந்த *unsupervised*-ல் வெறும் உள்ளீடுகள் மட்டுமே கொடுக்கப்படும். எத்தனை வகையில் பிரிக்க வேண்டும் என்பதோ, அவற்றின் எல்லைகள் என்ன என்பதோ கொடுக்கப்படாது. இது போன்ற *clustering*-ல் எல்லைகள் *K-means* மூலமாக கணக்கிடப்படுகின்றன. எவ்வளவு வகைகளில் பிரிக்க வேண்டும் என்பதை *elbow method*-மூலம் கணக்கிடலாம். அதாவது ஒரு வரையறையைக் கொடுத்து கற்கச் சொல்லுவது *supervised* என்றால், எவ்வித வரையறையும் இல்லாமல் கற்கச் சொல்லுவது *unsupervised* ஆகும்.

கீழ்க்கண்ட உதாரணத்தில் *X1, X2* எனும் இரண்டு அம்சங்கள்(*features*) கொடுக்கப்பட்டுள்ளன. *Y* என்று எதுவும் இல்லை. அதாவது வெறும் உள்ளீட்டுக்கான தரவுகளைக் கொண்டு மட்டுமே நாமாகவே பல்வேறு குழுக்களில் அவற்றை வகைப்படுத்திக் கொடுக்க வேண்டும்.

x1 = [15, 19, 15, 5, 13, 17, 15, 12, 8, 6, 9, 13]
x2 = [13, 16, 17, 6, 17, 14, 15, 13, 7, 6, 10, 12]

இதற்கான நிரல் மற்றும் விளக்கம் பின்வருமாறு.

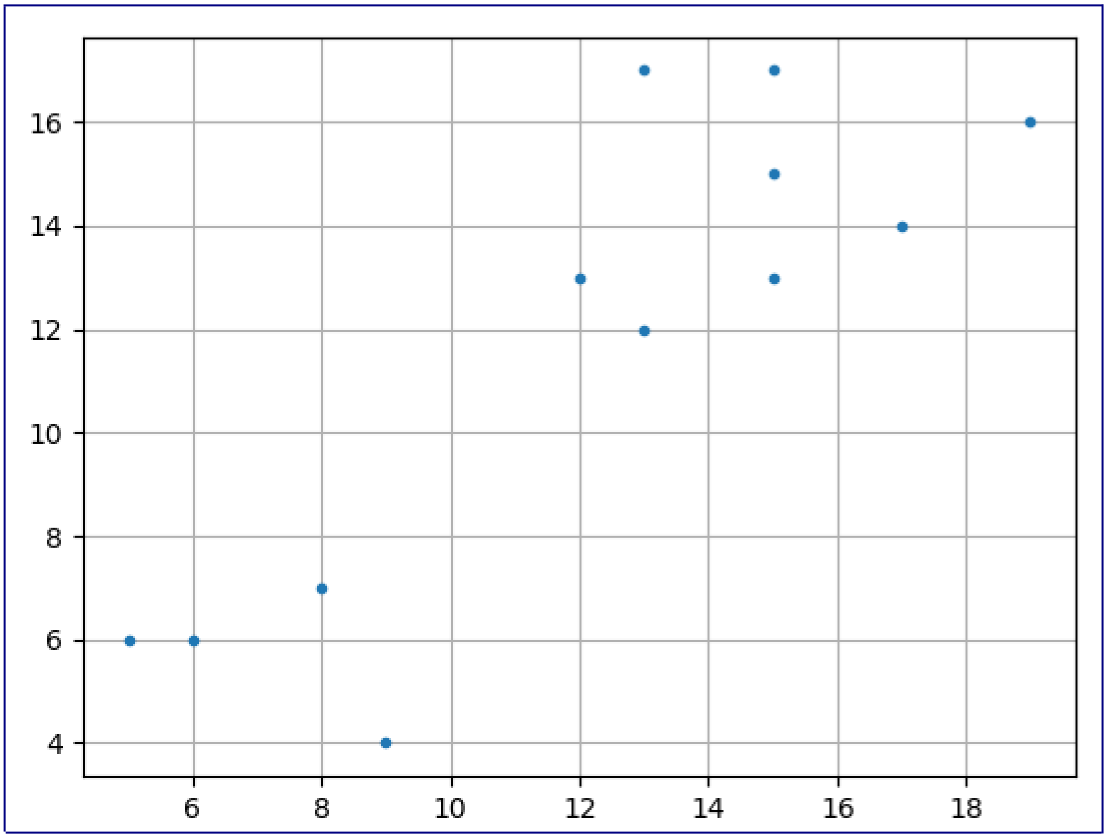
https://gist.github.com/nithyadurai87/185e332ebce7028af265adbe86db40d5

```
import matplotlib.pyplot as plt
import math

def plots(cluster1_x1,cluster1_x2,cluster2_x1,cluster2_x2):
    plt.figure()
    plt.plot(cluster1_x1,cluster1_x2,'.')
    plt.plot(cluster2_x1,cluster2_x2,'*')
    plt.grid(True)
    plt.show()

def round1(c1_x1,c1_x2,c2_x1,c2_x2):
    cluster1_x1 = []
    cluster1_x2 = []
    cluster2_x1 = []
    cluster2_x2 = []

    for i,j in zip(x1,x2):
      a = math.sqrt(((i-c1_x1)**2 + (j-c1_x2)**2))
      b = math.sqrt(((i-c2_x1)**2 + (j-c2_x2)**2))
      if a < b:
            cluster1_x1.append(i)
            cluster1_x2.append(j)
      else:
            cluster2_x1.append(i)
            cluster2_x2.append(j)

    plots(cluster1_x1,cluster1_x2,cluster2_x1,cluster2_x2)

    c1_x1 = sum(cluster1_x1)/len(cluster1_x1)
    c1_x2 = sum(cluster1_x2)/len(cluster1_x2)
    c2_x1 = sum(cluster2_x1)/len(cluster2_x1)
    c2_x2 = sum(cluster2_x2)/len(cluster2_x2)

    round2 (c1_x1,c1_x2,c2_x1,c2_x2)

def round2(c1_x1,c1_x2,c2_x1,c2_x2):
    cluster1_x1 = []
    cluster1_x2 = []
    cluster2_x1 = []
    cluster2_x2 = []

    for i,j in zip(x1,x2):
      c = math.sqrt(((i-c1_x1)**2 + (j-c1_x2)**2))
      d = math.sqrt(((i-c2_x1)**2 + (j-c2_x2)**2))
      if c < d:
            cluster1_x1.append(i)
            cluster1_x2.append(j)
      else:
            cluster2_x1.append(i)
            cluster2_x2.append(j)

    plots(cluster1_x1,cluster1_x2,cluster2_x1,cluster2_x2)

x1 = [15, 19, 15, 5, 13, 17, 15, 12, 8, 6, 9, 13]
x2 = [13, 16, 17, 6, 17, 14, 15, 13, 7, 6, 10, 12]

plots(x1,x2,[],[])
round1(x1[4],x2[4],x1[10],x2[10])
```

முதலில் X1, X2 எனும் இரண்டு அம்சங்களும் எவ்வாறு அமைந்துள்ளன என்பதை scatter plot மூலம் காணலாம். இன்னும் இரண்டாவது கொத்தில் என்னென்ன அம்சங்களை அமைக்க வேண்டும்

என்பது கண்டறியப்படவில்லை. எனவே அவை காலிப் பட்டியலாக அனுப்பப்படுகின்றன.

plots(x1,x2,[],[])



Centroids (திணிவுக்கான புள்ளி)

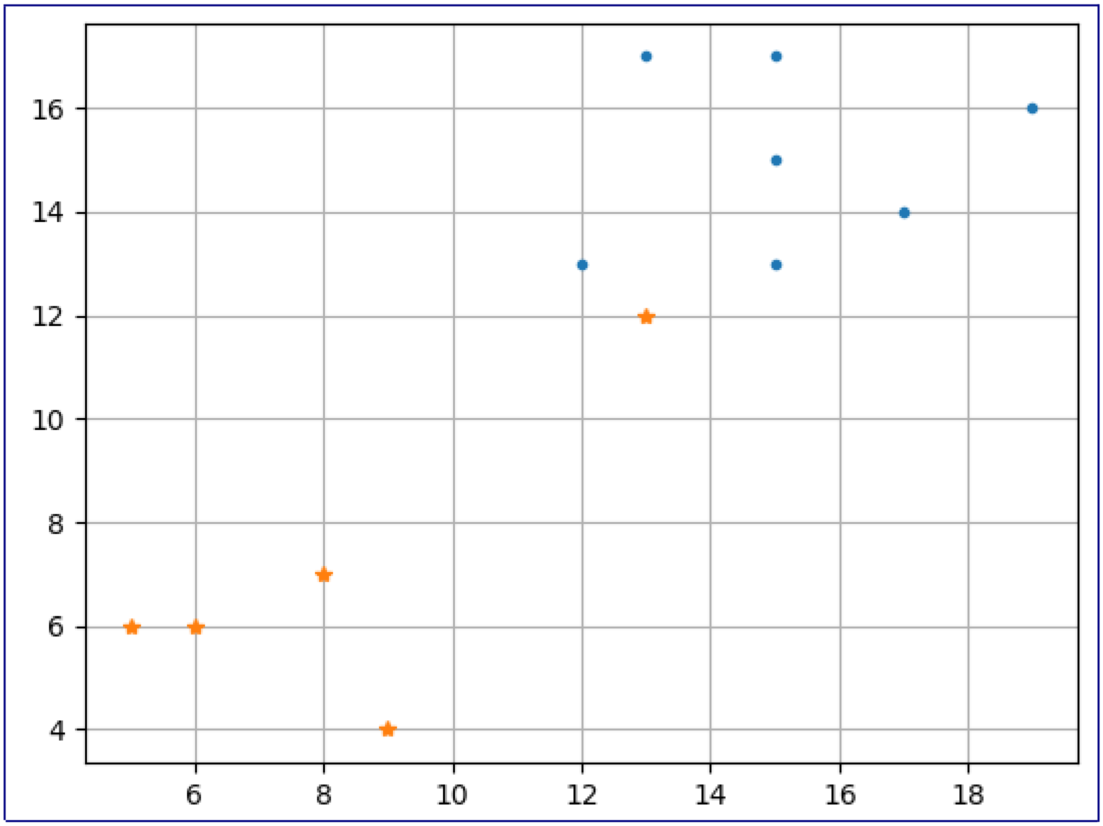
இரண்டு clusters-ஐ உருவாக்குவதற்கு முதலில் X1-லிருந்து இரண்டு எண்களையும், X2-லிருந்து இரண்டு எண்களையும் random-ஆக தேர்வு செய்ய வேண்டும். முதல் கொத்துக்கு X1-லிருந்து 13-ஐயும், X2-லிருந்து 17-ஐயும் தேர்வு செய்துள்ளோம். அவ்வாறே இரண்டாவது கொத்துக்கு X1-லிருந்து 9-ஐயும், X2-லிருந்து 17-ஐயும் தேர்வு செய்துள்ளோம். இவையே திணிப்புக்கான புள்ளிகள் (centroids) என்றழைக்கப்படுகின்றன. அதாவது இவற்றை அடிப்படையாக வைத்தே அனைத்தையும் நாம் இரண்டு கொத்தாகப் பிரிக்கப் போகிறோம். எனவே இரண்டு அம்சங்களில் உள்ள ஒவ்வொரு தரவுகளுக்கும், தேர்ந்தெடுக்கப்பட்ட இரண்டு திணிப்புப் புள்ளிகளுக்குமான தூரம் கீழ்க்கண்ட வாய்ப்பாடு மூலம் கணக்கிடப்படுகிறது.

*தூரம்1 = (x1_data - 13)**2 + (x2_data - 17)**2*
*தூரம்2 = (x1_data - 9)**2 + (x2_data - 10)**2*

x1	x2	தூரம்1	தூரம்2
15	13	(15-13)**2 + (13-17)**2 = 4 + 16 = 20 Sqrt(20) = 4.47	(15-9)**2 + (13-10)**2 = 36 + 9 = 45 Sqrt(45) = 6.70
19	16	(19-13)**2 + (16-17)**2 = 36 + 1 = 37 Sqrt(37) = 6.08	(19-9)**2 + (16-10)**2 = 100 + 36 = 136 Sqrt(136) = 11.66
15	17	2	9.21
5	6	13.6	5.65
13	17	0	8
17	14	5	8.94
15	15	2.82	7.81
12	13	4.12	4.24
8	7	11.18	3.16
6	6	13.03	5
9	4	8.06	0
13	12	5	4.47

இந்த இரண்டு கொத்துக்களில் முதல் கொத்தினுடைய தூரம் குறைவாக இருந்தால் அந்தப் புள்ளிகள் முதல் கொத்திலும், இல்லையெனில் இரண்டாவது கொத்திலும் அமைக்கின்றன. இவை முறையே மஞ்சள் மற்றும் ஊதா நிறத்தில் மேற்கண்ட படத்தில் காட்டப்பட்டுள்ளது. இவ்வாறாக முதல் கொத்துக்கான *x1, x2* மற்றும் இரண்டாவது கொத்துக்கான *x1, x2* என்று *4* அம்சங்கள் கணக்கிடப்படுகின்றன. அவை முறையே புள்ளி வடிவிலும், நட்சத்திர வடிவிலும் வரைபடமாக வரைந்து காட்டப்படுகின்றன.

```
cluster1_x1 = [15, 19, 15, 13, 17, 15, 12]
cluster1_x2 = [13, 16, 17, 17, 14, 15, 13]
cluster2_x1 = [5, 8, 6, 9, 13]
cluster2_x2 = [6, 7, 6, 10, 12]

plots(cluster1_x1,cluster1_x2,cluster2_x1,cluster2_x2)
```



இவ்வாறாக முதலில் இரண்டு கொத்துக்கள் உருவாக்கப்பட்ட பின்னர், அவற்றிலிருந்து மீண்டும் இரண்டு திணிப்புப் புள்ளிகள் தேர்ந்தெடுக்கப்படுகின்றன. ஆனால் இம்முறை இவை *random*-ஆக தேர்வு செய்யப்படுவதில்லை. இரண்டு கொத்துக்களிலும் அமைந்துள்ள *x1, x2*-க்கான *mean* கணக்கிடப்பட்டு அவையே திணிப்புப் புள்ளிகளாக அமைகின்றன. எனவே இன்னும் சற்று துல்லியமான இரண்டு கொத்துக்களை நாம் உருவாக்க முடியும்.

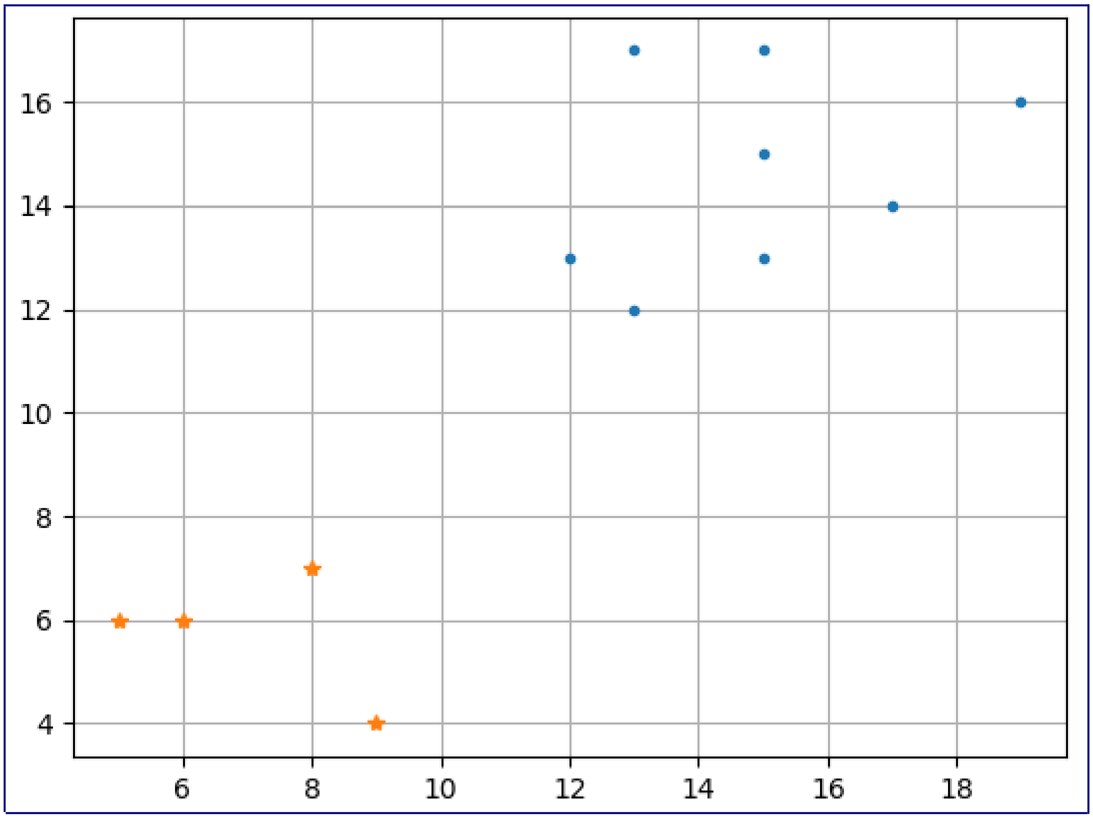
c1_x1 = (15 + 19 + 15 + 13 + 17 + 15 + 12) / 7
= 106 / 7
= 15.14

c1_x2 = [13 + 16 + 17 + 17 + 14 + 15 + 13] / 7
= 105 / 7
= 15

c2_x1 = [5 + 8 + 6 + 9 + 13] / 5
= 41 / 5
= 8.2

c2_x2 = [6 + 7 + 6 + 10+ 12] / 5
= 41 / 5
= 8.2

பின்னர் மீண்டும் ஒவ்வொரு *data*-க்கும், கண்டறிந்த திணிப்புப் புள்ளிகளுக்குமான தூரம் கணக்கிடப்படுகிறது.அதில் குறைவான அளவு தூரம் கொண்ட தரவுகள் அவற்றுக்கான கொத்தில் இணைகின்றன. இவ்வாறாக இங்கு மீண்டும் இரண்டு கொத்துகள் உருவாக்கப்படுகிறது. இவை தரவுகளை இன்னும் சற்று துல்லியமாகப் பிரிப்பதைக் காணலாம்.



இவ்வாறாகத் தரவுகள் தனக்குரிய கொத்தில் சரிவரப் பொருந்தும் வரையிலும், இதனையே நாம் தொடர்ச்சியாகச் செய்து கொண்டே செல்லலாம். இதுவே *clustering with k-means* எனப்படுகிறது. இதில் *k* என்பது எத்தனை கொத்துகள்/குழுக்கள் உருவாக்கப்பட வேண்டும் என்பதையும், *means* என்பது ஒவ்வொரு *features*-வுடைய சராசரியையும் கண்டுபிடித்து அதனடிப்படையில் குழுக்களை உருவாக்குவதையும் குறிப்பிடுகிறது. அடுத்ததாக இந்த *k*-ன் மதிப்பினை எவ்வாறு கணக்கிடுவது என்று பார்க்கலாம்.

Elbow Method

இது கொடுக்கப்பட்ட தரவுகளுக்கு எத்தனை குழுக்களை உருவாக்கினால் சரியாக இருக்கும் என்பதை ஒரு வரைபடம் மூலம் கண்டறிய உதவுகிறது. மேற்கண்ட அதே தரவுகளை இங்கும் நாம் பயன்படுத்திக் கொள்ளலாம். 2 குழுக்கள் என்பதை இது நமக்கு சரியாகக் காட்டுகிறதா எனப் பார்க்கலாம். இதற்கான நிரல் மற்றும் விளக்கம் பின்வருமாறு.

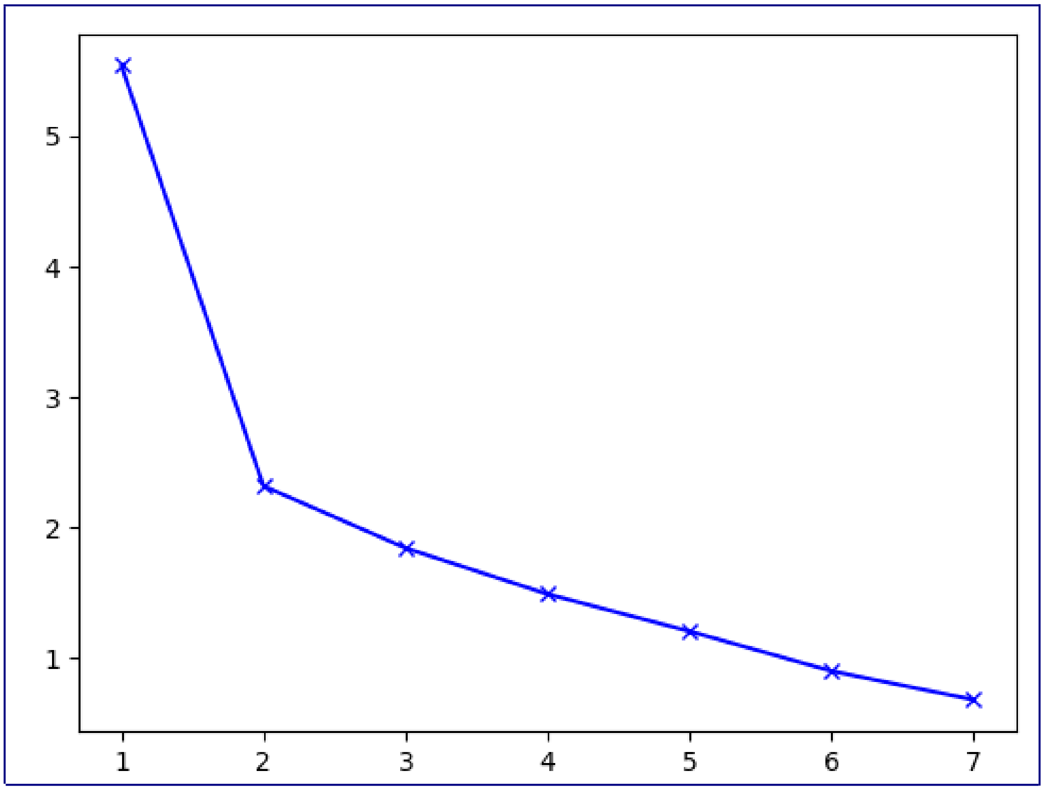
https://gist.github.com/nithyadurai87/10b5b273151c80be97579d684279cd84

```
from sklearn.cluster import KMeans
from sklearn import metrics
from scipy.spatial.distance import cdist
import numpy as np
import matplotlib.pyplot as plt

x1 = [15, 19, 15, 5, 13, 17, 15, 12, 8, 6, 9, 13]
x2 = [13, 16, 17, 6, 17, 14, 15, 13, 7, 6, 10, 12]
```

```
X = np.array(list(zip(x1, x2)))

distortions = []
K = range(1,8)
for i in K:
    model = KMeans(n_clusters=i)
    model.fit(X)
    distortions.append(sum(np.min(cdist(X, model.cluster_centers_, 'euclidean'),
axis=1)) / X.shape[0])

plt.plot()
plt.plot(K, distortions, 'bx-')
plt.show()
```

இதில் x1, x2 எனும் இரண்டு அம்சங்களும் numpy மூலம் x எனும் ஒரே அணியாக மாற்றப்படுகிறது. பின்னர் இத்தரவுகளைக் கொண்டு kmeans-க்குப் பயிற்சி அளிக்கப்படுகிறது. இப்பயிற்சியானது 1 முதல் 7 வரை பல்வேறு எண்ணிக்கையில் குழுக்களை அமைத்து பயிற்சி அளிக்கிறது. ஒவ்வொரு முறையும் அதன் தரவுகளுக்கும், திணிவுப் புள்ளிக்குமான விலகல் எவ்வளவு தூரம் இருக்கிறது என்பதைக் கணக்கிடுகிறது. இவ்வாறாக எந்த எண்ணிக்கையில் குழுக்களை அமைக்கும் போது அவற்றிலுள்ள தரவுகளின் விலகல் குறைகிறது என்பது கண்டுபிடிக்கப்படுகிறது. இந்த விலகல் மதிப்பே cost / distortion என்று அழைக்கப்படுகிறது.

பின்னர் இவை ஒரு வரைபடமாக வரையப்படுகின்றன. இதன் x அச்சில் குழுக்களின் எண்ணிக்கையும், y அச்சில் அதன் விலகல் மதிப்புகளும் அமைகின்றன. எனவேதான் ஒரே ஒரு கொத்தில் அனைத்துத் தரவுகளையும் அமைக்கும்போது அதனுடைய centroid-லிருந்து மற்ற தரவுகளின் விலகல் மதிப்பு 5-க்கு மேல் காட்டுவதையும், அதுவே 7 தனித்தனி கொத்துக்களாகப் பிரிக்கும்போது, அதனுடைய விலகல் மதிப்பு 1-க்குக் கீழ் காட்டுவதையும் காணலாம். இந்த வரைபடம் பார்ப்பதற்கு ஒரு முழங்கை வடிவில் இருப்பதால், இது Elbow method என்று அழைக்கப்படுகிறது. இந்த வரைபடத்தின் x-அச்சில் 2 என்ற புள்ளியில் அந்த முழங்கை போன்ற வடிவம் மடங்கி விரிவதால், அந்த எண்ணிக்கையில் தரவுகளைப் பிரித்தால் போதும் என்பதை நாம் தெரிந்து கொள்ளலாம். ஏனெனில் இதற்கு மேல் செல்லச் செல்ல விலகல் மதிப்புகள் ஓரளவுக்கே குறைகின்றன. இந்த புள்ளியில் தான் அந்த முழங்கை மடங்கும் நிலை ஏற்படுகிறது. எனவே தரவுகளை 2 குழுக்களில் பிரித்தால் சரியாக இருக்கும் என்பது கண்டுபிடிக்கப்படுகிறது..



silhouette_coefficient

ஒரு *algorithm*-ன் செயல்திறன் என்பது அது எவ்வளவு தூரம் சரியாகக் கணித்துள்ளது என்பதைப் பொறுத்தே அமைகிறது. இதுவரை நாம் கண்ட அனைத்திலும், *algorithm*-ன் கணிப்புகளை உண்மையான மதிப்புகளுடன் ஒப்பிட்டு அதன் செயல்திறனைக் கண்டறிந்தோம். ஆனால் *k-means* போன்ற *unsupervised learning*-ல் ஒப்பிடுவதற்கு நம்மிடம் தரவுகள் ஏதும் இல்லாத காரணத்தால், இதனைக் கண்டுபிடிக்க உதவும் ஒரு வழிமுறையே *silhouette_coefficient* ஆகும்.

அதாவது *k-means* முறையில் வகைப்படுத்தப்படும் தரவுகள், சரியான முறையில்தான் வகைப்படுத்தப்பட்டுள்ளதா எனக் கண்டறிய ஏற்கனவே *distortion* என்ற ஒன்றை அளவிட்டோம். இது ஒவ்வொரு தரவும் அதன் திணிவுப் புள்ளியிலிருந்து எவ்வளவு தூரம் விலகியிருக்கிறது என்பதை வைத்து, *kmeans*-ன் செயல்திறனைக் கணக்கிடுகிறது. அதுபோலவே இந்த *silhouette_coefficient* என்பது பின்வரும் வாய்ப்பாடு மூலம் தரவுகள் அமைந்துள்ள ஒவ்வொரு குழுவும் எவ்வளவு கச்சிதமாகப் பிரிக்கப்பட்டுள்ளது என்பதைக் கணக்கிடுகிறது.

ba / max(a,b)

இதில் *a* என்பது ஒரே குழுவில் உள்ள தரவுகளுக்கிடையேயான சராசரி தூரம். *b* என்பது ஒரு குழுவிற்கும் அதற்கடுத்த குழுவிற்கும் இடையே உள்ள தரவுகளுக்கிடையேயான சராசரி தூரம்.

கீழ்க்கண்ட எடுத்துக்காட்டில் நமது தரவுகள், *kmeans* மூலம் முதலில் *2* குழுக்களாகப் பிரிக்கப்படுகின்றன. அவ்வாறே *for loop* மூலம் அடுத்தடுத்து *3,4,5* மற்றும் *8* குழுக்களாகப் பிரிக்கப்படுகின்றன. இந்த *loop*-க்குள் குழுக்கள் கொடுக்கப்பட்ட எண்ணிக்கையில் ஒவ்வொரு முறை

அமையும்போதும், அது தரவுகளைப் பிரிக்கும் விதத்தை வரைபடமாக வரைந்து காட்டுகிறது மற்றும் அதன் *silhouette_coefficient* மதிப்பை வெளிப்படுத்துகிறது.

https://gist.github.com/nithyadurai87/f5f043df412b6e3c8291d0080422bd92

```
import numpy as np
from sklearn.cluster import KMeans
from sklearn import metrics
import matplotlib.pyplot as plt
plt.subplot(3, 2, 1)

x1 = [15, 19, 15, 5, 13, 17, 15, 12, 8, 6, 9, 13]
x2 = [13, 16, 17, 6, 17, 14, 15, 13, 7, 6, 10, 12]
plt.scatter(x1, x2)

X = np.array(list(zip(x1, x2)))

c = ['b', 'g', 'r', 'c', 'm', 'y', 'k', 'b']
m = ['o', 's', 'D', 'v', '^', 'p', '*', '+']

p = 1
for i in [2, 3, 4, 5, 8]:
      p += 1
      plt.subplot(3, 2, p)
      model = KMeans(n_clusters=i).fit(X)
      print (model.labels_)
      for i, j in enumerate(model.labels_):
          plt.plot(x1[i], x2[i], color=c[j], marker=m[j],ls='None')
      print (metrics.silhouette_score(X, model.labels_ ,metric='euclidean'))
plt.show()
```

print (model.labels_) என்பது முதல் குழுவை *0* என்றும் இரண்டாவது குழுவை *1* என்றும் குறிப்பிடுகிறது. எனவே *x1* மற்றும் *x2*-ல் உள்ள *12* தரவுகளும் எந்தெந்த குழுக்களில் சேர்க்கப்பட்டுள்ளன என்பதும் அதன் *coefficient* மதிப்பும் பின்வருமாறு வெளிப்படுகிறது.

[1 1 1 0 1 1 1 1 0 0 0 1]
0.6366488776743281

அவ்வாறே *3* குழுக்களாகப் பிரிக்கும்போது *0* முதல் குழுவையும், *1* இரண்டாவது குழுவையும், *2* மூன்றாவது குழுவையும் பின்வருமாறு குறிப்பிடுகிறது.

[0 0 0 1 0 0 0 2 1 1 1 2]
0.38024538066050284

இதுபோன்றே *4,5* மற்றும் *8* அளவில் குழுக்காளாகப் பிரிக்கும்போது தரவுகள் சேர்ந்துள்ள குழுக்களின் மதிப்பும், அக்குழுவிற்கான செயல்திறன் மதிப்பும் பின்வருமாறு வெளிப்படுகின்றன. இதை வைத்துப் பார்க்கும்போது *2* குழுக்களாகப் பிரிக்கும் போது மட்டுமே, இது அதிக அளவு செயல் திறனை *(0.63)* வெளிப்படுத்துவதைக் காணலாம்.

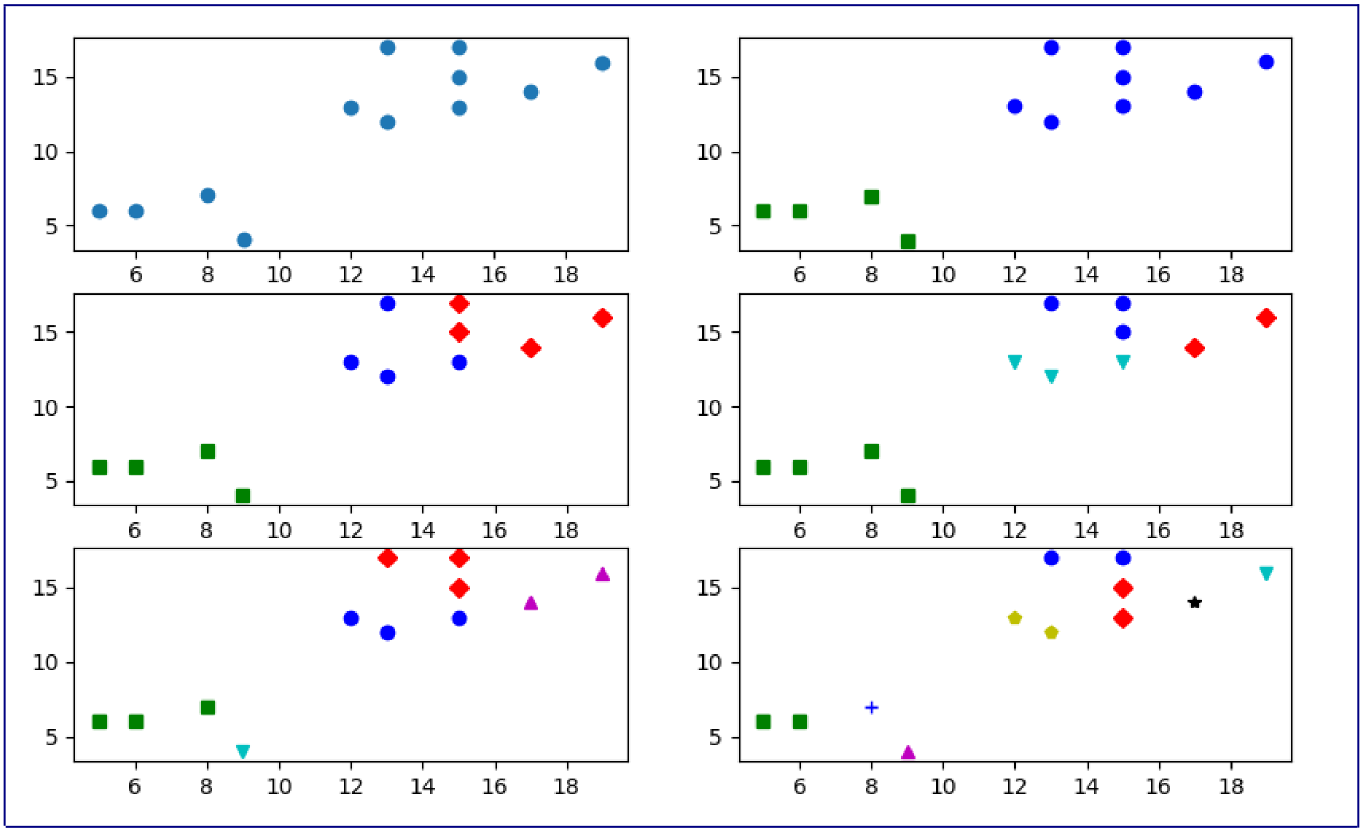
[2 0 0 1 0 0 0 2 1 1 3 2]
0.32248773306926665

[2 4 0 1 0 4 0 2 1 1 3 2]
0.38043265897525885

[6 7 3 4 3 1 6 2 0 4 5 2]
0.27672998081717154

கீழ்க்கண்ட வரைப்படத்தில் முதலாவதாக உள்ளது வெறும் தரவுகளுக்கான படம். இரண்டாவதாக உள்ளது *2* குழுக்காளாகப் பிரிக்கும்போது வெளிப்படும் வரைபடம். அடுத்தடுத்து உள்ளது

3,4,5,8 எண்ணிக்கையில் குழுக்களை அமைக்கும்போது வெளிப்படுகின்ற வரைபடங்கள். அதிகபட்சமாக 8 குழுக்கள் வரை தரவுகள் பிரிக்கப்படுகின்றன. எனவே ஒவ்வொரு குழுவிலும் உள்ள தரவுகளை வித்தியாசப்படுத்திக் காட்ட, 8 நிற வண்ணங்களும் 8 வெவ்வேறு வடிவங்களும் கொண்ட இரண்டு பட்டியல் உருவாக்கப்படுகிறது. அவை ஒவ்வொன்றாக loop-க்குள் சென்று பின்வருமாறு வெளிப்படுகின்றன.



SVM

Support Vector Machine (SVM) என்பது தரவுகளை வகைப்படுத்திப் பிரிப்பதற்கான ஒரு வழிமுறை ஆகும். ஏற்கெனவே இதற்கென logistic regression என்பதைப் பற்றிப் பார்த்தோம். ஆனால் இந்த SVM என்பது வகைப்படுத்துதல் எனும் வேலையை logistic-ஐ விட இன்னும் சற்று துல்லியமாக அமைக்கிறது. நேர்கோடு மூலம் பிரிக்கப்படும் தரவுகளுக்கு large margin classifier எவ்வாறு உதவுகிறது என்பதையும், நேர்கோடு முறையில் பிரிக்கப்பட முடியாத தரவுகளுக்கு kernels எவ்வாறு உதவுகிறது என்பதையும் இப்பகுதியில் காணலாம்.

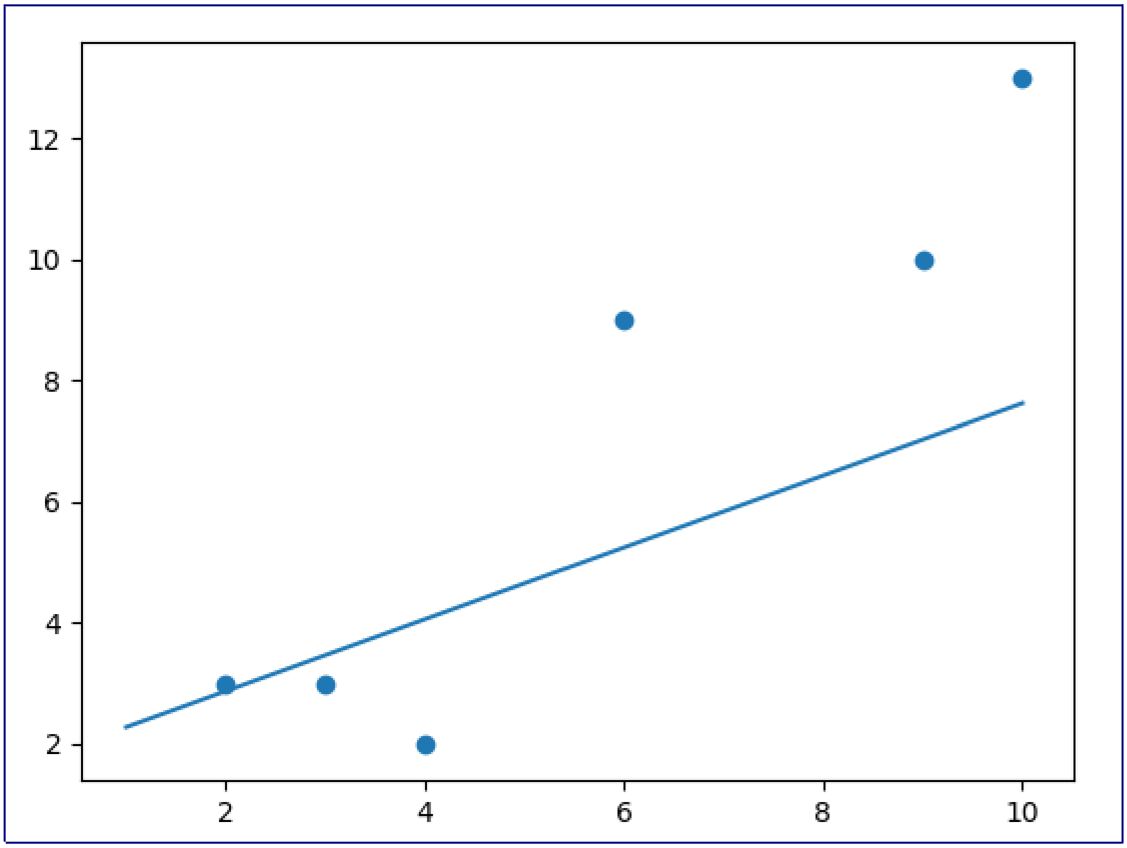
Large margin classifier (linear)

கீழ்க்கண்ட உதாரணத்தில் ஒரு நேர்கோடு மூலம் வகைப்படுத்த முடியும் தரவுகளை logistic எவ்வாறு பிரிக்கிறது, svm எவ்வாறு பிரிக்கிறது என்பதைக் காட்டியுள்ளோம். இதில் x1, x2 எனும் இரண்டு அம்சங்கள் உள்ளன. அவை 2 பரிமாணங்கள் (2 dimension matrix) கொண்ட ஒரே அணியாக numpy மூலம் மாற்றப்படுகின்றன. பின்னர் அத்தரவுகளைக் கொண்டு logistic-க்கும், svm-க்கும் பயிற்சி அளிக்கிறோம். பின்னர் ஒவ்வொன்றும் தரவுகளைப் பிரிப்பதற்கான நேர்கோட்டினை சரியாக எங்கு அமைக்கின்றன என்பதைக் காண்பதற்கான நிரல் classifier()-க்குள் எழுதப்பட்டுள்ளது.

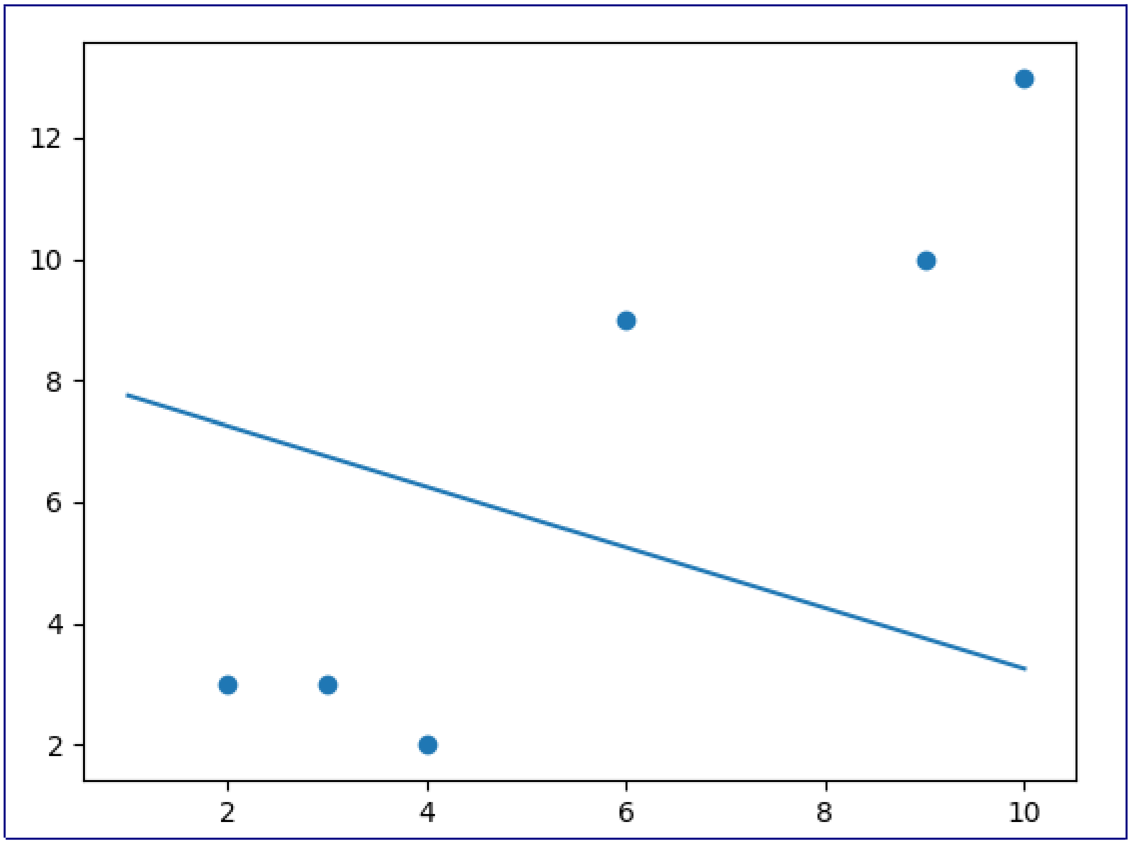
https://gist.github.com/nithyadurai87/2de5a6a6f7cc03c2791305f5c33d43d7

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm
from sklearn.linear_model.logistic import LogisticRegression

def classifier():
    xx = np.linspace(1,10)
    yy = -regressor.coef_[0][0] / regressor.coef_[0][1] * xx -
regressor.intercept_[0] / regressor.coef_[0][1]
    plt.plot(xx, yy)
    plt.scatter(x1,x2)
    plt.show()

x1 = [2,6,3,9,4,10]
x2 = [3,9,3,10,2,13]

X = np.array([[2,3],[6,9],[3,3],[9,10],[4,2],[10,13]])
y = [0,1,0,1,0,1]

regressor = LogisticRegression()
regressor.fit(X,y)
classifier()

regressor = svm.SVC(kernel='linear',C = 1.0)
regressor.fit(X,y)
classifier()
```

logistic மூலம் தரவுகள் பிரிக்கப்படும்போது அதற்கான நேர்கோடு பின்வருமாறு அமைகிறது. அதாவது கீழே உள்ள வகைக்கு மிகவும் நெருக்கமாக எவ்வித இடைவெளியும் இல்லாமல் நேர்கோடு அமைக்கப்பட்டுள்ளது. ஆனால் மேலே உள்ள வகைக்கும் கோட்டிற்குமான இடைவெளியோ மிகவும் அதிகமாக உள்ளது.



SVM மூலம் தரவுகள் பிரிக்கப்படும்போது இரண்டு வகைக்கும் நடுவில் உள்ள கோடு அவ்விரண்டு வகையிலிருந்தும் சமமான அளவு தூரத்தில் உள்ளது. எனவே தான் இது equal margin / large margin classifier என்று அழைக்கப்படுகிறது. இது logistic regression-க்கான ஒரு optimization-ஆகவே கருதப்படுகிறது.



Kernels (non-linear)

Kernel என்பது நேர்கோடு போட்டு பிரிக்க முடியாத சற்று கடினமான *non-linear* முறையில் அமைந்துள்ள தரவுகளை வகைப்படுத்துவதற்குப் பயன்படுகிறது. இது போன்ற நேர்கோட்டில் பொருந்தாத தரவுகளைப் பொறுத்துவதற்கு ஏற்கெனவே *polynomial regression* என்ற ஒன்றைப் பார்த்தோம். ஆனால் அதில் ஒவ்வொரு *features*-வுடைய *higher order* மதிப்புகள் கணக்கிடப்பட்டு அவை புதிதாக இணைந்துள்ள அம்சங்களாகக் கணக்கில் கொள்ளப்படும். எனவே தரவுகள் முழுதாகப் பொருந்தும் வரையிலும் *square, cube* என்று அடுத்தடுத்த *order*-ல் *features*-ஐக் கணக்கிட்டு இணைத்துக் கொண்டே செல்வோம். இவ்வாறு செய்யும்போது பயிற்சி அளிக்கப்படும் தரவில் அதிக அளவு அம்சங்கள் சேர்க்கப்படுவதால், ஒரு *algorithm* கற்றுக் கொள்வதற்கான நேரமும் கணினி அனைத்தையும் அதிக அளவில் நினைவில் வைத்துக் கொள்ள வேண்டிய தேவையும் அதிகரிக்கிறது. இதைத் தவிர்ப்பதற்காக வந்ததே *kernels / similarity functions* ஆகும்.

இது புதிது புதிதாக அம்சங்களை இணைக்காமல், ஏற்கெனவே உள்ள அம்சங்களில் இருந்து புதிய அம்சங்களைக் கணக்கிட்டுப் பயன்படுத்துகிறது. உதாரணத்துக்கு நமது பயிற்சித் தரவில் 5 அம்சங்களும் 100 மாதிரித் தரவுகளும் உள்ளன என்று வைத்துக்கொள்வோம். *Polynomial* எனும் போது இத்தகைய 5 *features*-க்கும் *square* மற்றும் *cube* மதிப்புகள் கண்டுபிடிக்கப்பட்டு, கடைசியில் அவை 20 -க்கும் மேலான *features*-ஆக வந்து நிற்கும். அதுவே *kernel* மூலம் பொறுத்தும் போது ஒவ்வொரு அம்சங்களிலும் உள்ள 100 மாதிரிகளில் இருந்து ஒரு தரவினை தேர்வு செய்து அதனை *landmark*-ஆக அமைக்கிறது. பின்னர் அதிலிருந்து மற்ற தரவுகள் எவ்வளவு தூரத்தில் அமைந்துள்ளன என்பது கணக்கிடப்படுகிறது. அவை *landmark*-க்கு அருகில் இருந்தால் 1 எனவும், இல்லையெனில் 0 எனவும் வகைப்படுத்தப்படுகின்றன. இதை வைத்தே புதிய *feature* கணக்கிடப்படுகிறது. அதாவது பயிற்சித் தரவில் உள்ள 5 அம்சங்களுக்கு வெறும் 5 புதிய *features* மட்டுமே இம்முறையில்

கணக்கிடப்படுகின்றன.

இந்த similarity function-க்கான சமன்பாடு பின்வருமாறு. இதுவே kernel என்றும் அழைக்கப்படுகிறது. இந்த kernel இக்கணக்கீடுகளை நிகழ்த்துவதற்கு பல்வேறு வாய்ப்பாடுகளைப் பெற்றிருக்கும். அதில் ஒன்றான exp()-க்கான சமன்பாடு கீழே கொடுக்கப்பட்டுள்ளது. இதுவே gaussian kernel என்று அழைக்கப்படுகிறது. இதே போன்று polynomial kernel, string kernel, chi-squared kernel, histogram-intersection kernel என்று பல்வேறு வகையான வாய்ப்பாடுகள் kernel-ல் உள்ளன.

f1 = similarity (x ,l1)
*= exp (-(||x-l||**2 / 2*sigma squared))*

SVM without kernels என்பது logistic regression-ஐக் குறிக்கிறது. அதாவது kernels மூலம் உருவாக்கப்பட்ட புதிய features-ஐப் பயன்படுத்தாமல், நேரடியாக raw feature-ஐக் கொண்டு மட்டுமே வகைப்படுத்துதல் நிகழ்ந்தால், அது logistic regression-ஐயே குறிக்கிறது. எனவே எப்போது kernel-ஐப் பயன்படுத்தலாம் எப்போது logistic-ஐப் பயன்படுத்தலாம் என்று பார்ப்போம். தேர்ந்தெடுக்கப்பட்ட அம்சங்களின் எண்ணிக்கை(100000 or 100), பயிற்சிக்கு அளிக்கப்பட்ட மாதிரித் தரவுகளின் எண்ணிக்கையை(10000) விட மிகவும் அதிகமாக இருந்தாலோ அல்லது மிகவும் குறைவாக இருந்தாலோ svm without kernel-ஐப் பயன்படுத்தலாம். அதுவே features-ன் எண்ணிக்கை(1000) மிகவும் அதிகமாக இல்லாமல் ஓரளவுக்கு சற்று அதிகமாக இருக்கும்போது svm with kernel-ஐப் பயன்படுத்தலாம்.

கீழ்க்கண்ட எடுத்துக்காட்டில் பல்வேறு அம்சங்களை வைத்து ஒரு மலர் மல்லியா, ரோஜாவா, தாமரையா என்று வகைப்படுத்தப்படுகிறது. இவை svm without kernel அதாவது logistic மூலம் வகைப்படுத்தப்படுவதைவிட kernel மூலம் வகைப்படுத்தப்படும்போது அதன் accuracy அதிகரிப்பதைக் காணலாம்.

https://gist.github.com/nithyadurai87/9d7cc99cc4ae18a3707cc76f8711193b

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm
import pandas as pd
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.linear_model.logistic import LogisticRegression
from matplotlib.colors import ListedColormap

df = pd.read_csv('./flowers.csv')
X = df[list(df.columns)[:-1]]
y = df['Flower']
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 0)

logistic = LogisticRegression()
logistic.fit(X_train, y_train)
y_pred = logistic.predict(X_test)
print ('Accuracy-logistic:', accuracy_score(y_test, y_pred))

gaussian = SVC(kernel='rbf')
gaussian.fit(X_train, y_train)
y_pred = gaussian.predict(X_test)
print ('Accuracy-svm:', accuracy_score(y_test, y_pred))
```

வெளியீடு:

Accuracy-logistic: 0.868421052631579
Accuracy-svm: 0.9736842105263158

PCA

Principle Component Analysis என்பது அதிக அளவு பரிமாணங்கள் கொண்ட தரவுகளை குறைந்த அளவு பரிமாணங்கள் கொண்டதாக மாற்றுவதற்குப் பயன்படுகிறது. எடுத்துக்காட்டாக 1000 அம்சங்களைக் கொண்டு ஒரு விஷயம் கணிக்கப்படுகிறது என வைத்துக் கொள்வோம். PCA-ஆனது இந்த 1000 X-ஐ 100 X-ஆகவோ அல்லது இன்னும் குறைந்த பரிமாணங்கள் கொண்டதாகவோ மாற்றிக் கொடுக்கும். அதாவது Y எண்ணிக்கையைப் பற்றிக் கவலைப்படாது. வெறும் X எண்ணிக்கையை மட்டும் குறைக்கும். எனவேதான் PCA என்பது dimensionality reduction-க்கு உதவுகின்ற ஒரு சிறப்புவகை வழிமுறை ஆகும். இதன் செயல்பாடுகளில் உள்ள படிகள் பின்வருமாறு.

- முதலில் பயிற்சித் தரவுகளைப் பெற்றுக் கொள்ளுதல் (x1,y1),(x2,y2),(x3,y3)...
- அடுத்ததாக PCA மூலம் பயிற்சித் தரவில் உள்ள x அனைத்தையும் நமக்குத் தேவையான அளவு குறைந்த எண்ணிக்கையில் மாற்றுதல்
- பின்னர் குறைக்கப்பட்ட புதிய x -ஐக் கொண்டு பயிற்சி அளித்தல்

பொதுவாக இந்த PCA அனைத்து இடத்திலும் பயன்படாது. சற்று அரிதாகவே பயன்படும். எடுத்துக்காட்டுக்கு மனித முகங்கள் அல்லது ஊர்திகள் போன்றவற்றை அடையாளப்படுத்தும் algorithm-க்கு பயிற்சி அளிக்கப்படும் தரவுகளில் குறைந்தபட்சம் 1 லட்சம் features-ஆவது இருக்கும். ஏனெனில் ஒரு ஊர்தியின் சக்கரம், கைப்பிடி, இருக்கை, பக்கக் கண்ணாடிகள், முன் விளக்குகள் என்று ஒவ்வொரு சின்னச் சின்ன விஷயங்களையும் அடையாளப்படுத்த அதிக அளவில் features அமைந்திருக்கும். இதுபோன்ற இடங்களில் , அவை அனைத்தையும் பயன்படுத்தாமல் குறைந்த அளவில் features-ஐ மாற்றுவதற்கு PCA பயன்படுகிறது. எப்போதும் pca-ஐப் பயன்படுத்துவதற்கு முன்பு feature scaling என்ற ஒன்று கண்டிப்பாக நடைபெற வேண்டும். இதுவே data-preprocessing என்று அழைக்கப்படும்.

கீழ்க்கண்ட எடுத்துக்காட்டில் நாம் புரிந்து கொள்ளச் சுலபமாக இருக்க வேண்டும் என்பதற்காக 4 dimension கொண்ட தரவுகள் 2 dimension-ஆக PCA மூலம் மாற்றப்பட்டுள்ளது. PCA பயன்படுத்துவதற்கு முன்னர் StandardScalar மூலம் தரவுகள் normalize செய்யப்படுகின்றன. பின்னர் ஒரு மலர் மல்லியா, ரோஜாவா, தாமரையா என்று தீர்மானிக்க அவ்விதழ்களுடைய நீள அகலமும், அவற்றின் மேற்புற இதழ்களுடைய நீள அகலமுமாக 4 அம்சங்கள் உள்ளன. இவை PCA மூலம் x1, x2 எனும் இரண்டு அம்சங்களாக மாற்றப்படுகின்றன. இவ்விரண்டு அம்சங்களின் அடிப்படையில் அமையும் 3 வகை மலர்களும் 3 நிறங்களில் வரைபடமாக வரைந்து காட்டப்பட்டுள்ளது.

https://gist.github.com/nithyadurai87/20d18bbda53e43de19222e24d330a398

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

df = pd.read_csv('./flowers.csv')
X = df[list(df.columns)[:-1]]
y = df['Flower']
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 0)

pca = PCA(n_components=2)
```

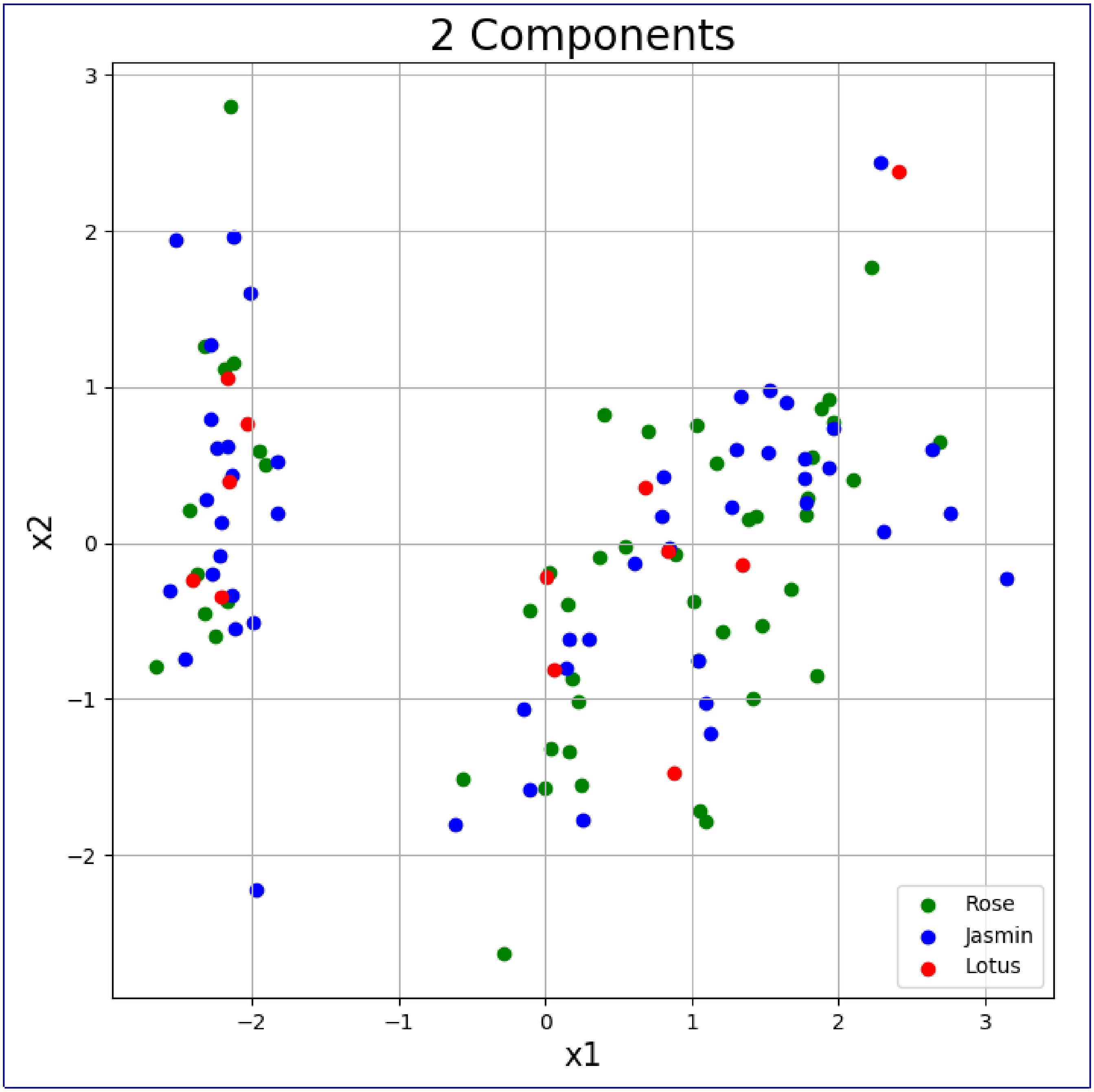
```
x = StandardScaler().fit_transform(X_train)
new_x = pd.DataFrame(data = pca.fit_transform(x), columns = ['x1', 'x2'])

df2 = pd.concat([new_x, df[['Flower']]], axis = 1)

fig = plt.figure(figsize = (8,8))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('x1', fontsize = 15)
ax.set_ylabel('x2', fontsize = 15)
ax.set_title('2 Components', fontsize = 20)
for i, j in zip(['Rose', 'Jasmin', 'Lotus'],['g', 'b', 'r']):
    ax.scatter(df2.loc[df2['Flower'] == i, 'x1'], df2.loc[df2['Flower'] == i,
'x2'], c = j)
ax.legend(['Rose', 'Jasmin', 'Lotus'])
ax.grid()
plt.show()

print (pca.explained_variance_ratio_)

print (df.columns)
print (df2.columns)
```

வெளியீடு :

[0.72207932 0.24134489]

Index(['Sepal_length', 'Sepal_width', 'Petal_length', 'Petal_width', 'Flower'], dtype='object')

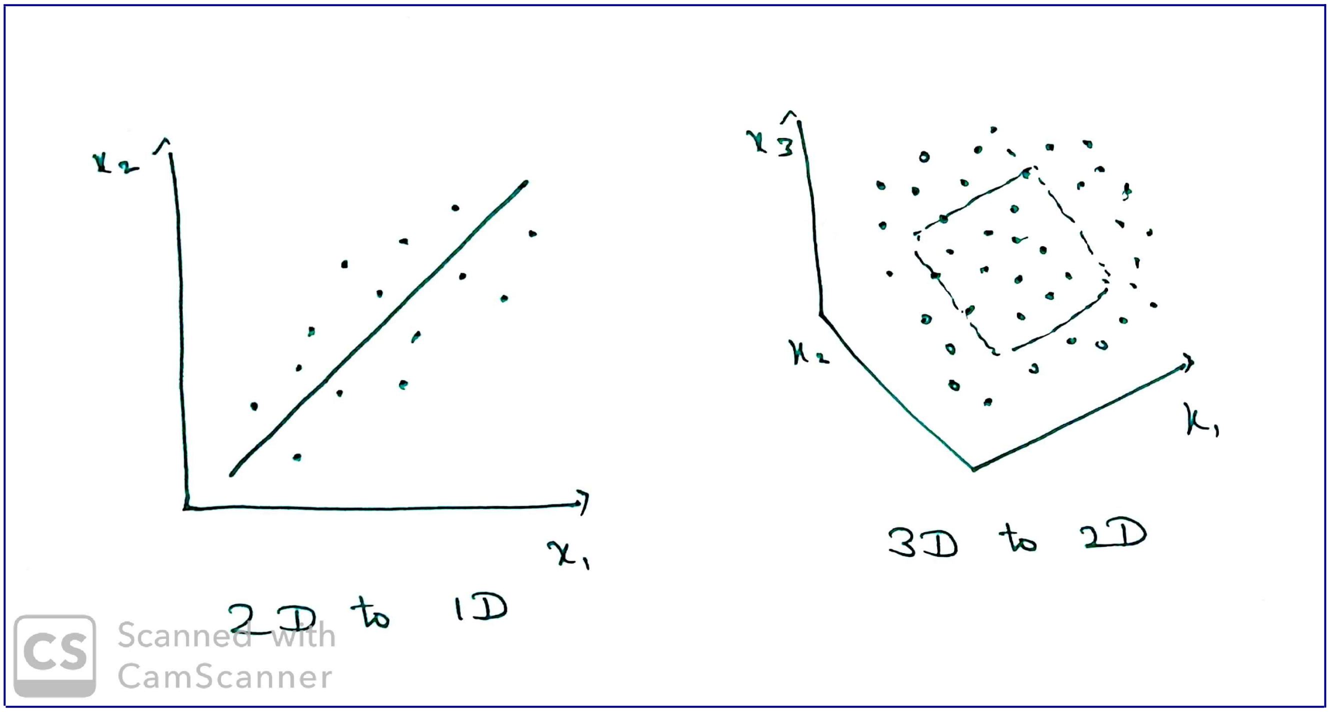
Index(['x1', 'x2', 'Flower'], dtype='object')



இதனுடைய வெளியீட்டில் என்பது *explained variance* என்பது *[0.72207932 0.24134489]* என வந்துள்ளது. இவ்விரண்டு மதிப்புகளையும் கூட்டினால் *0.96* என்று வரும். இதற்கு என்ன அர்த்தம் என்றால் இவ்விரண்டு *components*-ம் சேர்ந்து *96%* தகவல்களை உள்ளடக்கியுள்ளது என்று அர்த்தம். ஏனெனில் *features*-ஐக் குறைக்கும்போது தகவல் இழப்பு ஏற்பட வாய்ப்பு உள்ளது. எனவே *variance* என்பது எவ்வளவு சதவீதம் தகவல்கள் ஒவ்வொன்றிலும் சேமிக்கப்பட்டுள்ளன என்பதைக் கூற உதவுகிறது. இதைப்பற்றியும், *PCA* செயல்படும் விதத்தையும் இன்னும் விளக்கமாகக் கீழே காணலாம்.

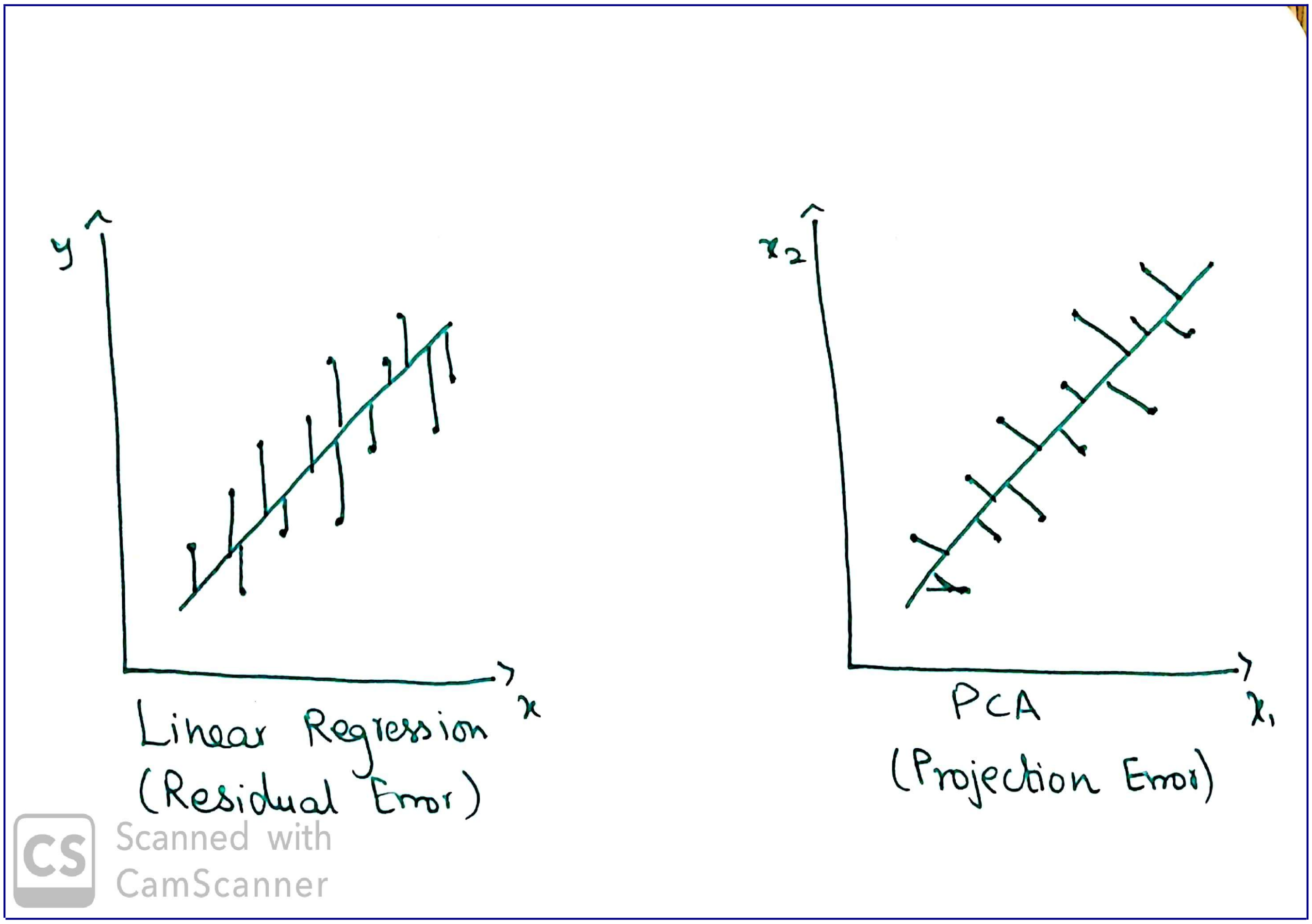
Data Projection

தரவுகளின் பரிமாணங்களை குறைப்பதற்கு உதவும் திட்டமே Projection line அல்லது projection area எனப்படுகிறது. கீழ்க்கண்ட வரைபடங்களை கவனிக்கவும். இடதுபுறம் உள்ள படத்தில் 2 dimension கொண்ட தரவுகள் 1 dimension ஆக மாற்றப்படுவதற்கான திட்டம் உள்ளது. இதில் x1, x2 எனும் 2 அம்சங்களுக்கான scatter plot உள்ளது. அவற்றின் நடுவில் அமைந்துள்ள கோடுதான் projection-க்கான திசை ஆகும். இத்திசையை நோக்கியே தரவுகள் அனைத்தும் சென்று ஒரே பரிமாணம் கொண்டதாக மாற்றப்படுகின்றன. அவ்வாறே வலப்புறம் உள்ள படத்தில் x1, x2, x3 எனும் 3 அம்சங்களுக்கான தரவுகள் உள்ளது. அவற்றிற்கான projection area-ஆனது 2 பரிமாணங்களைக் கொண்டு வரைபடத்தில் காணப்படுவது போன்று அடையாளப்படுத்தப்படுகிறது. சுற்றியுள்ள தரவுகள் அனைத்தும் அப்பரப்பளவு கொண்ட பகுதிக்குள் சென்று 2 பரிமாணங்கள் கொண்ட வெக்டராக மாற்றப்படுகின்றன.



Projection Error

மேற்கண்ட இரண்டு படங்களிலும் தரவுகள் அவை அமைந்துள்ள இடத்திற்கும், project செய்யப்பட்ட இடத்திற்குமான இடைவெளியே projection error என்று அழைக்கப்படுகிறது. 2d-ஐ 1d-ஆக மாற்றுவதற்கான படத்தைப் பார்க்கும்போது உங்களுக்கு linear regression நினைவுக்கு வரலாம். ஆனால் PCA என்பது linear regression அல்ல. ஏனெனில் நடுவில் உள்ள அக்கோடு prediction-க்குப் பயன்படாது. வெறும் projection-க்கு மட்டுமே பயன்படுகிறது. அவ்வாறே அக்கோட்டினை வைத்து Y மதிப்புகளை கணித்துச் சொல்லாது. வெறும் x மதிப்புகளை இடமாற்றம் செய்வதற்கே இக்கோடு பயன்படுகிறது. மேலும் linear regression-ல் sum of squares error என்பது இடைப்பட்ட தூரத்தை செங்குத்தாகக் கணக்கிடுகிறது. ஆனால் PCA-ல் projection error என்பது பக்கவாட்டில் கணக்கிடப்படுகிறது. இது பின்வருமாறு.



Scanned with CamScanner

Compressed components

அதிக அளவு கொண்ட பரிமாணங்கள் எவ்வாறு சிறிய அளவில் சுருக்கப்படுகிறது, அதில் உள்ள படிகள் என்னென்ன என்று பின்வருமாறு பார்க்கலாம்.

1. முதலில் தரவுகள் அனைத்தும் feature scaling செய்யப்பட வேண்டும். இதுவே data preprocessing என்று அழைக்கப்படுகிறது.(x1, x2, x3...xn)

2. அடுத்து features-க்கிடையேயான தரவுகள் எவ்வாறு அமைந்துள்ளன என்பதைக் காண covariance matrix உருவாக்கப்படுகிறது. இதற்கான வாய்ப்பாடு பின்வருமாறு. இதுவே sigma என்று அழைக்கப்படுகிறது.

covariance matrix / sigma = (1/m).summation of(1 to m)[x . transpose of x]

இந்த அணியானது symmetric positive definite எனும் பண்பு கொண்டுள்ளதா எனப் பார்க்க வேண்டும். அப்போதுதான் இதை வைத்து projection-க்கான வெக்டரை உருவாக்க முடியும்.

3. svd() அல்லது eig() எனும் function-ஐப் பயன்படுத்தி projection-க்கான வெக்டரை உருவாக்கலாம். இவை முறையே single value decomposition என்றும், eigenvector என்றும் அழைக்கப்படும். இது பின்வருமாறு

[u,s,v] = svd(sigma)

இது 3 அணிகளை உருவாக்கும். u என்பதுதான் projection -க்கான அணி. அதாவது u1, u2, u3...un வரை இருக்கும். இதிலிருந்து நமக்கு வேண்டிய அளவு features-ஐத் தேர்வு செய்யலாம். அதாவது u1, u2, u3...uk - இதில் k என்பது எவ்வளவு principle components என்பதைக் குறிக்கிறது. இதற்கான வாய்ப்பாடு பின்வருமாறு.

principle components = transpose of (u[: , 1:k]).x

4. அடுத்ததாக எவ்வளவு principle components இருந்தால் தகவல் இழப்பு எதுவும் இருக்காது என்பதைக் கண்டுபிடிக்க வேண்டும். இதைக் கண்டுபிடித்துக் கூறுவதே variance என்று அழைக்கப்படும். பொதுவாக 99% variance அளவில் இருக்குமாறு பார்த்துக் கொண்டால் நல்லது. எனவே k-ன் மதிப்பைக் கண்டுபிடிக்கும் வாய்ப்பாடானது பின்வருமாறு அமைகிறது.
Average squared projection error / Total variance in the data >= 0.99 (எனவே 99% அளவு variance-ஐ தக்க வைத்துக் கொள்கிறது)

Where,
Avg. squared projection error = (1/m).summation of(1 to m) . square of (x - projected x)
Total variance in the data = (1/m).summation of(1 to m) . square of (x)

k-ன் மதிப்பை ஒவ்வொன்றாக அதிகரித்து மேற்கண்ட வாய்ப்பாட்டில் பொருத்தி எப்போது அதன் மதிப்பு 0.99 ஐத் தாண்டுகிறது எனப் பார்ப்பது ஒரு வகை. இதற்கு பதிலாக svd()-யிலிருந்து பெறுகின்ற S அணியை பின்வரும் வாய்ப்பாட்டில் பொருத்தி k -ன் மதிப்பை நேரடியாகக் கண்டுபிடிக்கலாம்.
summation of(1 to k) S[i,j] / summation of(1 to m) S[i,j] >= 0.99

இவ்வாறாக அதிக அளவு கொண்ட பரிமாணங்கள் தகவல் இழப்பு எதுவும் நடைபெறாமல் குறைந்த அளவில் சுருக்கப்படுகிறது..

Neural Networks

மனிதனுடைய மூளை எவ்வாறு கற்கிறது என்பதை முன்னோடியாகக் கொண்டு உருவாக்கப்பட்டதே Neural network ஆகும். முதலில் குழந்தையாகப் பிறக்கும்போது மனித மூளைக்கு ஒன்றுமே தெரியாது. பின்னர் அதிலுள்ள ஒரு மூளை நரம்பு (நியூரான்) புதிய விஷயத்தைக் கற்றுக் கொள்ளத் தொடங்குகிறது. அடுத்ததாக மற்றொரு நரம்பு ஏற்கெனவே கற்றுக் கொண்டுள்ள விஷயத்தோடு சேர்த்து இன்னொரு புதிய விஷயத்தையும் கற்றுக் கொள்கிறது. இவ்வாறே பல்வேறு நரம்புகள் வலைப்பின்னல் வடிவில் ஒன்றோடொன்று பிணைக்கப்பட்டு தொடர்ச்சியாக பல்வேறு புதுப்புது விஷயங்களைக் கற்றுக் கொண்டே வருகின்றன. இதை அடிப்படையாக வைத்து உருவாக்கப்பட்டதே Neural Network ஆகும்.

இது ஒவ்வொரு விஷயத்தையும் வகைப்படுத்தி வகைப்படுத்திக் கற்கிறது. எனவே இதன் சூத்திரம் classification problem-ஐ ஒத்திருக்கும். binary classification -ல் x1, x2 என்று இரண்டுfeatures-இருக்கிறதெனில், logistic - ஆனது அதனை நேரடியாக எடுத்துக் கொண்டு h(x) -ஐ கணிக்கும். ஆனால் neural network-ஆனது raw features-ஐப் பயன்படுத்தாமல் தனக்கென ஒரு hidden layer-ஐ உருவாக்கிக் கொண்டு, அதில் பல activation units-ஐ உருவாக்கிக் கணிக்கிறது. இதற்கான சூத்திரம் பின்வருமாறு.

$$h(x) = \frac{1}{1 + e^{-\theta^T x}}$$

$$= g(\theta_0 a_0 + \theta_1 a_1 + \theta_2 a_2)$$

Where

$$a_0 = g(\theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2)$$

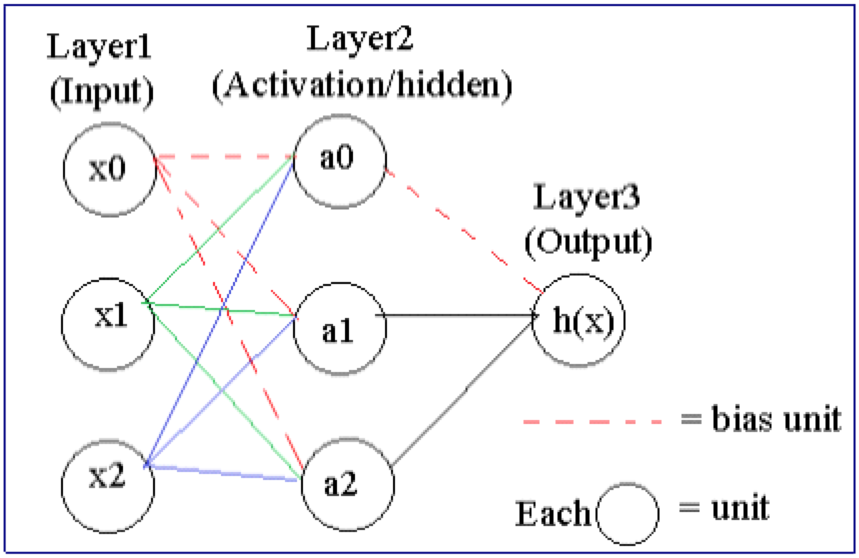
Likewise a_1 & a_2

activation unit-ன் மதிப்பானது 0 முதல் 1 வரை அமைவதால், sigmoid function-க்குள் அதனுடைய parameters மற்றும் features அமைகிறது. இதை வைத்தே முதல்activation unit-ன் மதிப்பு கணக்கிடப்படுகிறது. இவ்வாறே ஒவ்வொருactivation unit-ன் மதிப்புகளும் கணக்கிடப்படுகின்றன. Parameters-ஐ தீட்டா என்று குறித்தோம் அல்லவா, Neural networks-ல் இவை weights என்று அழைக்கப்படுகின்றன. எனவே கடைசியாக கணிக்கப்படும் h(x) மதிப்புகள், அதனுடைய activation units மற்றும் weights ஐ இணைத்து sigmoid function-ஆல் கணிக்கப்படுகின்றன.

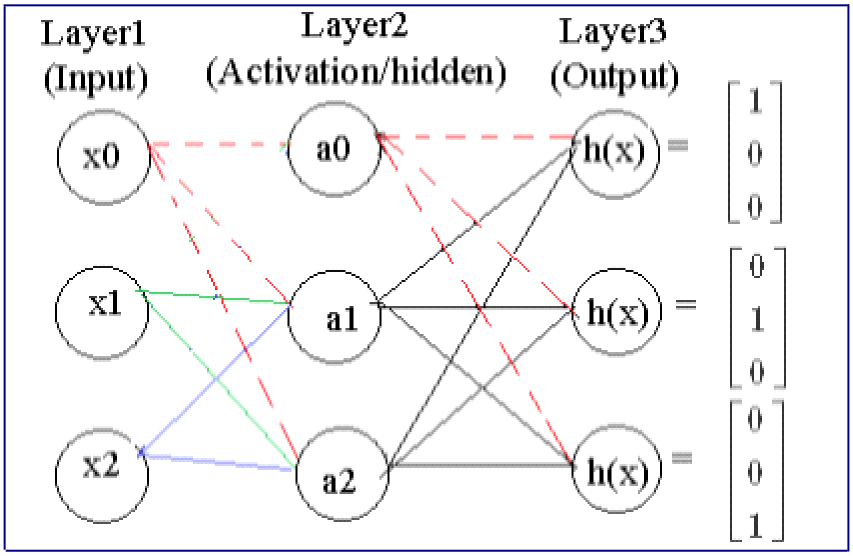
Neural Network அமைப்பு

கணிப்புக்குத் தேவையான features-ன் எண்ணிக்கை மிகவும் அதிகமாக இருக்கும்போது logistic-க்குப் பதிலாக நாம் neural networks-ஐப் பயன்படுத்தலாம்.

Binary classification-க்கான neural network பின்வருமாறு அமையும்.



Multi-class classification-க்கான neural network பின்வருமாறு அமையும்.



அணிகளின் பெருக்கலுக்கு துணைபுரியும் வகையில் சேர்க்கப்படும் x0, a0 மதிப்புகள் bias units என்றழைக்கப்படுகின்றன.

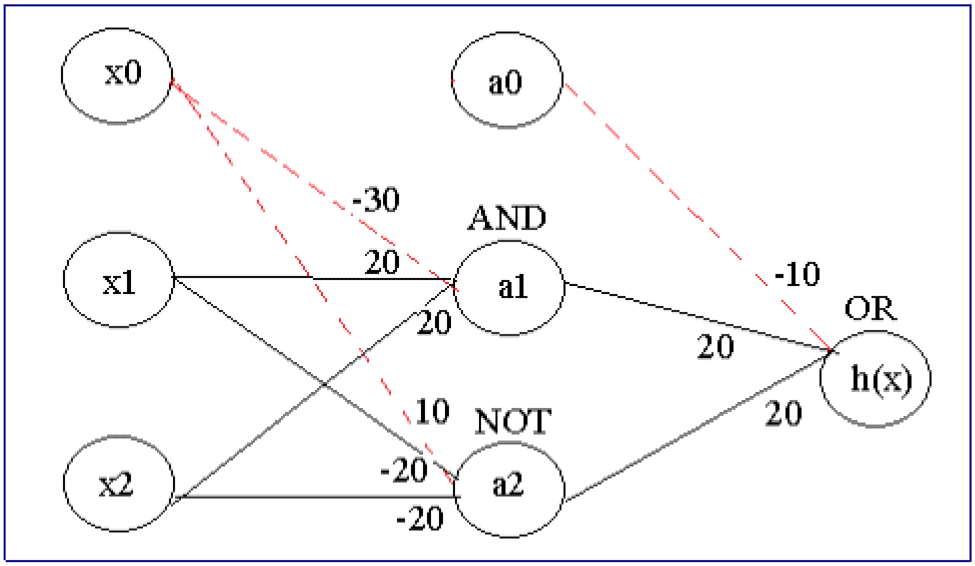
Input layer: மூல அம்சங்கள் முதலாவது அடுக்கில் காணப்படும்.

Output layer: கணிக்கப்படும் கணிப்புகள் கடைசி அடுக்கில் அமையும்.

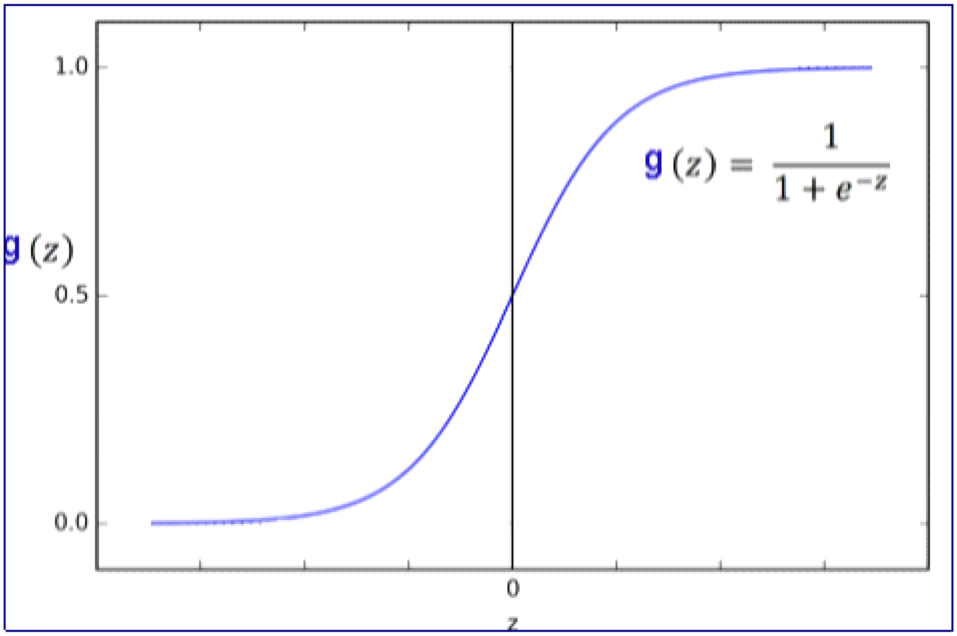
Hidden layer / Activation layer – இடையில் பல்வேறு மறைமுக அடுக்குகள் காணப்படும். முதல் மறைமுக அடுக்கில் மூல அம்சங்களை வைத்து உருவாக்கப்பட்ட செயல்படுத்தும் அலகுகள் (activation units) காணப்படும். அடுத்தடுத்த மறைமுக அடுக்கில் அடுத்தடுத்த செயல்படுத்தும் அலகுகள் காணப்படும்.

h(x) கணிப்புகள் நிகழும் விதம்

கீழ்க்கண்ட படத்தில் ஒவ்வொரு அலகுகளுக்குமான எடைகள் கொடுக்கப்பட்டுள்ளன. இவற்றை sigmoid சூத்திரத்தில் பொருத்தி ஒவ்வொரு அலகுக்குமான h(x) மதிப்பு கணக்கிடப்படுகிறது. எடைகளின் மதிப்பினைப் பொறுத்து இவைகளின் மதிப்பு AND, OR, NOT போன்ற விதிகளின் படி அமையும்.



எடுத்துக்காட்டுக்கு -30, 20, 20 எனும் மதிப்புகளை g(z) சூத்திரத்தில் பொருத்திப் பார்க்கவும். x1, x2 மதிப்புகள் 0,0 ஆக இருந்தால் என்னவரும்? 0,1 ஆக இருந்தால் என்னவரும்? 1,0 மற்றும் 1,1 மதிப்புகளுக்கு என்ன வரும்? போன்றவை கணக்கிடப்படுகிறது.



AND: 0,0 எனும்போது g(z) மதிப்பு -30 என எதிர்மறையில் அமைகிறது. மேற்கண்ட sigmoid வரைபடத்தில் -30 என்பது 0 என்பதைக் குறிக்கும். இவ்வாறே அடுத்தடுத்த மதிப்புகள் கணக்கிடப்படுகின்றன. இதற்கான அட்டவணை AND -க்கான truth table-ஐ ஒத்திருப்பதைக் காணலாம். அதாவது x0 மற்றும் x1 1-ஆக அமைந்தால் மட்டுமே h(x)=1 ஐ வெளிப்படுத்தும்.

Weights = -30,20,20			
x1	x2	h(x)	AND
0	0	h(x) = -30.x0 + 20.x1 + 20.x2 = -30 + 20.0 + 20.0 = -30	0
0	1	h(x) = -30.x0 + 20.x1 + 20.x2 = -30 + 20.0 + 20.1 = -10	0
1	0	h(x) = -30.x0 + 20.x1 + 20.x2 = -30 + 20.1 + 20.0 = -10	0
1	1	h(x) = -30.x0 + 20.x1 + 20.x2 = -30 + 20.1 + 20.1 = -30 + 40 = 10	1

OR: -10, 20, 20 எனும் மதிப்புகளை g(z) சூத்திரத்தில் பொருத்திப் பார்க்கவும். இதற்கான அட்டவணை OR -க்கான truth table-ஐ ஒத்திருப்பதைக் காணலாம். அதாவது x0 மற்றும் x1 1-ஆக அமைந்தால் மட்டுமே h(x)=1 ஐ வெளிப்படுத்தும். அதாவது x0 அல்லது x1 இரண்டில் ஏதாவது ஒன்று 1-ஆக அமைந்தால் கூட h(x)=1 ஐ வெளிப்படுத்தும்.

Weights = -10,20,20			
x1	x2	h(x)	OR
0	0	h(x) = -10.x0 + 20.x1 + 20.x2 = -10 + 20.0 + 20.0 = -10	0
0	1	h(x) = -10.x0 + 20.x1 + 20.x2 = -10 + 20.0 + 20.1 = -10 + 20 = 10	1
1	0	h(x) = -10.x0 + 20.x1 + 20.x2 = -10 + 20.1 + 20.0 = -10 +20 = 10	1
1	1	h(x) = -10.x0 + 20.x1 + 20.x2 = -10 + 20.1 + 20.1 = -10 + 40 = 30	1

NOT : இரண்டாவது அடுக்கில் உள்ள 3-வது அலகானது NOT x1 AND NOT X2 மூலம் கணக்கிடப்படுகிறது. அதாவது NOT x1 மற்றும் NOT x2 இரண்டின் மதிப்பும் AND -மூலம் மீண்டும் கணக்கிடப்படுகின்றன. இதற்கான எடைகள் 10, -20 என்று அமையும்.

Weights = 10,-20						
x1	x2	h(x1)	NOT(x1)	h(x2)	NOT(x2)	NOT x1 AND NOT X2
0	0	h(x) = 10.x0 -20.x1 = 10	1	h(x) = 10.x0 -20.x2 = 10	1	0
0	1	h(x) = 10.x0 -20.x1 = 10	1	h(x) = 10.x0 -20.x2 = -10	0	0
1	0	h(x) = 10.x0 -20.x1 = -10	0	h(x) = 10.x0 -20.x2 = 10	1	0
1	1	h(x) = 10.x0 -20.x1 = -10	0	h(x) = 10.x0 -20.x2 = -10	0	1

எனவே இவைகள் ஒன்றாக சேர்ந்து மேற்கண்ட *neural network*-க்கான மதிப்பு பின்வருமாறு அமையும்.

x1	x2	a1	a2	h(x)
0	0	0	1	1
0	1	0	0	0
1	0	0	0	0
1	1	1	0	1

Forward propagation

Layer 1 : $a = x$

$$\theta = \begin{bmatrix} \theta_0 & \theta_1 & \theta_2 \end{bmatrix} \quad x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix}$$

Layer 2 : $a = g(\theta . x)$

$$\theta = \begin{bmatrix} \theta_0 & \theta_1 & \theta_2 \end{bmatrix} \quad a = \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix}$$

Layer 3 : $h(x) = g(\theta . a)$

முதலாவது அடுக்கில் உள்ள செயல்படுத்தும் அலகானது *(activation unit)* அதன் மூல அம்சங்களாக *(raw features)*அமையும். இதுவே உள்ளீட்டுக்கான அடுக்கு ஆகும்.

இரண்டாவதாக உள்ளது மறைமுக அடுக்கு. இதில் உள்ள செயல்படுத்தும் அலகானது முதலாவதில் உள்ள அம்சங்கள் மற்றும் அதன் எடைகளைப்*(weights)* பொறுத்து அமையும்.

கடைசியாக உள்ளது வெளியீட்டுக்கான அடுக்கு ஆகும். இதில் உள்ள அலகானது மறைமுக அடுக்குகளில் உள்ள அலகுகள் மற்றும் அதன் எடைகளைப்*(weights)* பொறுத்து அமையும்.

இவ்வாறே ஒவ்வொரு அடுக்கிலும் உள்ள செயல்படுத்தும் அலகுகளின் மதிப்பும் அதனுடைய எடையும் சேர்ந்து அடுத்தடுத்த அடுக்குகளில் உள்ள அலகுகளின் மதிப்பை தீர்மானிப்பதே *forward*

propagation எனப்படும்.

Back propagation

நமது *neural network*-ல் உள்ள ஒவ்வொரு அலகுக்கும் என்னென்ன எடைகளைப் பயன்படுத்தினால், தவறுகளைக் குறைக்கலாம் எனக் கண்டுபிடிப்பதே *back propagation* ஆகும் . ஒவ்வொரு அடுக்கிலும் நிகழும் தவறைக் கண்டுபிடிக்க அதன் *partial derivative* மதிப்புகள் பின்னிருந்து முன்னாகக் கணக்கிடப்படுகின்றன. பின்னர் அவைகளை ஒன்று திரட்டி அந்த *network*-ன் *cost* கண்டுபிடிக்கப்படுகிறது. பொதுவாக *gradient descent algorithm* -ஆனது குறைந்த அளவு *cost* வெளிப்படக் கூடிய வகையில் *neuron*-களின் எடையை அமைக்க இந்த *back propagation* -ஐப் பயன்படுத்துகிறது.

delta = error of each node in the corresponding layer

Layer 3 : delta3 = h(x) – y

Layer 2 : delta2 = theeta T .delta3 . a .* 1-a*

Layer 1 : delta 1 = theeta T .delta2 . a .* 1-a*

where g'(z) = a . 1-a = This is g-prime. = derivative of the activation function g*

*. * = element-wise multiplication*

$$\frac{\partial}{\partial \theta_0} J = \frac{1}{m} \left(\text{Accumulator matrix} \right)$$

Perceptron

Perceptron என்பதே *neural networks*-க்கான அடிப்படை. இது ஒரு நேர்கோடு மூலம் பிரிக்க வல்ல தரவுகளுக்கான *binary classification algorithm* ஆகும். ஆனால் இது *logistic regression* போன்று தனது கற்றலை அமைக்காது. ஒரு நியூரான் எவ்வாறு கொஞ்சம் கொஞ்சமாக கற்றுக் கொள்கிறதோ அதனை அடிப்படையாக வைத்து, பயிற்சித் தரவுகளைப் பற்றிப் படிப்படியாகக் கற்றுக் கொள்கிறது. கீழ்க்கண்ட எடுத்துக்காட்டில் *4* பயிற்சித் தரவுகள் கொடுக்கப்பட்டுள்ளன. அதில் *x1, x2* எனும் *2 features*-ஐ வைத்து *0* அல்லது *1* எனும் வகையின் கீழ் அமையும் தரவுகள் பயிற்சிக்கு உள்ளன.

x1 , x2 , y
[0.4 ,0.3 ,1],
[0.6 ,0.8 ,1],
[0.7 ,0.5 ,1],
[0.9 ,0.2 ,0]

Neural Networks என்பது நேரடியாக கற்றுக் கொள்ளாமல் இடையில் பல *activation units*-ஐ உருவாக்கி அதனடிப்படையில் கற்றுக் கொள்ளும் என்று ஏற்கெனவே பார்த்தோம். இங்கும் *features*-ஐயும் அதனுடைய *weights*-ஐயும் இணைத்து நேரடியாக *hypothesis*-ஐக் கற்றுக் கொள்ளாமல், இடையில் *activation unit*-ஐக் கணக்கிடுகிறது. பின்னர் அம்மதிப்பின் அடிப்படையில் தரவுகளுக்கு ஏற்றார்

போன்று weights-ஐ மாற்றி சரியான முறையில் கற்றுக் கொள்கிறது. இது பின்வருமாறு. parameters என்பதே இங்கு weights என அழைக்கப்படுகிறது.

https://gist.github.com/nithyadurai87/e6794ec008a7855681db4ba9164b54af

```
def predict(row, weights):
    activation = weights[0]
    for i in range(len(row)-1):
        activation += weights[i + 1] * row[i]
    return 1.0 if activation > 0.0 else 0.0

def train_weights(dataset, l_rate, n_epoch):
    weights = [0.0 for i in range(len(dataset[0]))]
    for epoch in range(n_epoch):
        sum_error = 0.0
        for row in dataset:
            error = row[-1] - predict(row, weights)
            sum_error += error**2
            weights[0] = weights[0] + l_rate * error
            for i in range(len(row)-1):
                weights[i + 1] = weights[i + 1] + l_rate * error * row[i]
        print('epoch=%d, error=%.2f' % (epoch, sum_error))
    print (weights)

dataset = [[0.4,0.3,1],
      [0.6,0.8,1],
      [0.7,0.5,1],
      [0.9,0.2,0]]

l_rate = 0.1
n_epoch = 6
train_weights(dataset, l_rate, n_epoch)
```

நிரலுக்கான வெளியீடு:

epoch=0, error=2.00
epoch=1, error=2.00
epoch=2, error=2.00
epoch=3, error=2.00
epoch=4, error=1.00
epoch=5, error=0.00
[0.1, -0.16, 0.06999999999999998]

கணக்கீடுகள் நிகழும் விதம்:

முதலில் கொடுக்கப்பட்டுள்ள features-வுடன் இணைக்கப்பட வேண்டிய weights-ன் மதிப்பாக 0, 0, 0 என்பதை வைத்து தனது கற்றலைத் தொடங்குகிறது. முதலில் உள்ள பூஜ்ஜியம், x0 எனும் bias unit-க்கான மதிப்பாகும். இந்த bias unit எப்போதும் 1 எனும் மதிப்பையே பெற்றிருக்கும் என ஏற்கெனவே பார்த்தோம். அடுத்தடுத்து உள்ள பூஜ்ஜியங்கள் x1, x2 -க்கான weights மதிப்பாகும். இவற்றை வைத்து பின்வரும் வாய்ப்பாட்டின் மூலம் முதல் தரவுக்கான [0.4 ,0.3 ,1] activation unit கணக்கிடப்படுகிறது. இதுவே heaviside activation function என்று அழைக்கப்படுகிறது. sigmoid போன்று இது மற்றொரு வகை.
Activation_unit_1 = w0.x0 + w1.x1 + w2.x2
= 0(1) + 0(0.4) + 0(0.3)
= 0

if Activation_unit > 0, Predict 1
else   Predict 0.

இவ்வாறு கண்டறிந்த மதிப்பு, 0-ஐ விட அதிகமாக இருந்தால் 1 எனவும், இல்லையெனில் 0 எனவும் predict செய்யும். இங்கு 0 என predict செய்யும். ஆனால் பயிற்சித் தரவில் 1 என கொடுக்கப்பட்டுள்ளது. இவ்வாறு பயிற்சித் தரவில் உள்ள மதிப்பு, activation unit கணித்த மதிப்புடன் ஒத்துப் போகவில்லையெனில்(1 != 0) weights-ன் மதிப்பினை மாற்றி அடுத்த தரவுக்கு பயிற்சி அளிக்க வேண்டும். பின்வரும் வாய்ப்பாட்டின் மூலம் புதிய weights கணக்கிடப்படுகிறது.

w0 = w0 + learning_rate * (actual-predict) * x0

இதில் ஒவ்வொரு weight-ம் தன்னுடைய பழைய மதிப்புடன் learning rate-ஐக் கூட்டுகிறது. இந்த learning rate என்பது gradient descent-ல் நாம் பயன்படுத்துகின்ற மதிப்பினை ஒத்ததே ஆகும். அதாவது update-ன் அளவானது இந்த learning rate மூலம் கட்டுப்படுத்தப்படுகிறது. இதன் மதிப்பு 0.1 என வைக்கப்பட்டுள்ளது. அதாவது மிகச்சிறிய அளவில் இதனுடைய weights, adjust செய்யப்பட வேண்டும் என்பதையே இது குறிக்கிறது. பின்னர் இக்கூட்டுத் தொகையுடன் உண்மையான மதிப்புக்கும் - கணிப்புக்கும் உள்ள வேறுபாட்டின் மதிப்பும், weights இணைக்கப்பட்டுள்ள features-ன் மதிப்பும் பெருக்கப்படுகிறது. இவ்வாறாக புதிய weight-ன் மதிப்பு கணக்கிடப்படுகிறது.

இந்த வாய்ப்பாட்டைப் பயன்படுத்திக் கணக்கிடப்பட்ட weights-ன் மதிப்புகள் பின்வருமாறு.
w0 = 0 + 0.1 * 1 * 1 = 0.10
w1 = 0 + 0.1 * 1 * 0.4 = 0.04
w2 = 0 + 0.1 * 1 * 0.3 = 0.03

இத்தகைய புதிய weights-ஐப் பயன்படுத்தி 2-வது தரவுக்கான [0.6 ,0.8 ,1] activation unit பின்வருமாறு கணக்கிடப்படுகிறது.
Activation_unit_2 = w0.x0 + w1.x1 + w2.x2
= 0.1(1) + 0.04(0.6) + 0.03(0.8)
= 0.1 + 0.024 + 0.024
= 0.148

இங்கு 0-ஐ விட அதிகமாக இருப்பதால் 1 என predict செய்யும். பயிற்சித் தரவிலும் 1 என உள்ளது. ஆகவே weights-ஐ மாற்றாமல் 3-வது தரவுக்கான [0.7 ,0.5 ,1] activation unit கணக்கிடப்படுகிறது.
Activation_unit_3 = w0.x0 + w1.x1 + w2.x2
= 0.1(1) + 0.04(0.7) + 0.03(0.5)
= 0.1 + 0.028 + 0.015
= 0.143

இங்கும் 1 என predict செய்கிறது. பயிற்சித் தரவிலும் 1 என உள்ளது. ஆகவே weights-ஐ மாற்றாமல் 4-வது தரவுக்கான [0.9 ,0.2 ,0] activation unit கணக்கிடப்படுகிறது.
Activation_unit_4 = w0.x0 + w1.x1 + w2.x2
= 0.1(1) + 0.04(0.9) + 0.03(0.2)
= 0.1 + 0.036 + 0.006
= 0.142

w0 = 0.1 + 0.1 * -1 * 1 = 0.0
w1 = 0.04 + 0.1 * -1 * 0.9 = -0.05
w2 = 0.03 + 0.1 * -1 * 0.2 = 0.01

இங்கு 1 என கணிக்கிறது. ஆனால் உண்மையில் 0 என உள்ளது. எனவே மீண்டும் weights கணக்கிடப்படுகிறது. இவ்வாறாக கொடுக்கப்பட்டுள்ள 4 பயிற்சித் தரவுகளில் 2 சரியாக கணிக்கப்பட்டுள்ளது, 2 தவறாக கணிக்கப்பட்டுள்ளது. இத்துடன் முதல் epoch முடிகிறது. அதாவது ஒரு சுற்றில் அனைத்துப் பயிற்சித் தரவுகளும் மேற்கண்ட சோதனைக்கு உட்படுத்தப்பட்டு, algorithm கற்றுக் கொள்வதையே 1 epoch என்கிறோம். இது பின்வருமாறு.

Epoch = 0						
x1	x2	y	weights	activation units	predicted_y	updated_weight for next row If y != predicted_y
0.4	0.3	1	0,0,0	0*1 + 0*0.4 + 0*0.3 = 0	0	0 + 0.1 * 1 = 0.10 0 + 0.1 * 1 * 0.4 = 0.04 0 + 0.1 * 1 * 0.3 = 0.03
0.6	0.8	1	0.1, 0.04, 0.03	0.1*1 + 0.04*0.6 + 0.03*0.8 = 0.1 + 0.024 + 0.024 = 0.148	1	
0.7	0.5	1	0.1, 0.04, 0.03	0.1*1 + 0.04*0.7 + 0.03*0.5 = 0.1 + 0.028 + 0.015 = 0.143	1	
0.9	0.2	0	0.1, 0.04, 0.03	0.1*1 + 0.04*0.9 + 0.03*0.2 = 0.1 + 0.036 + 0.006 = 0.142	1	0.1 + 0.1 * -1 = 0.0 0.04 + 0.1 * -1 * 0.9 = -0.05 0.03 + 0.1 * -1 * 0.2 = 0.01

முதல் epoch-ன் கடைசியில் புதிதாக கணக்கிடப்பட்ட மதிப்புகளே அடுத்த epoch-ன் பயிற்சித் தரவுகளுடன் சேர்த்து பயன்படுத்தப்படுகிறது. இவ்வாறாக 6 முறை epochs கணக்கிடப்படுகிறது. இது பின்வருமாறு.

Epoch = 1						
x1	x2	y	weights	activation units	predicted_y	updated_weight for next row If y != predicted_y
0.4	0.3	1	0, -0.05, 0.01	0*1 + -0.05*0.4 + 0.01*0.3 = 0 + -0.02 + 0.003 = -0.017	0	0 + 0.1 * 1 = 0.10 -0.05 + 0.1 * 1 * 0.4 = -0.01 0.01 + 0.1 * 1 * 0.3 = 0.04
0.6	0.8	1	0.1, -0.01, 0.04	0.1*1 + -0.01*0.6 + 0.04*0.8 = 0.1 + -0.006 + 0.032 = 0.126	1	
0.7	0.5	1	0.1, -0.01, 0.04	0.1*1 + -0.01*0.7 + 0.04*0.5 = 0.1 + -0.07 + 0.02 = 0.1	1	
0.9	0.2	0	0.1, -0.01, 0.04	0.1*1 + -0.01*0.9 + 0.04*0.2 = 0.1 + -0.009 + 0.008 = 0.1	1	0.1 + 0.1 * -1 = 0.0 -0.01 + 0.1 * -1 * 0.9 = -0.1 0.04 + 0.1 * -1 * 0.2 = 0.02
Epoch = 2						
x1	x2	y	weights	activation units	predicted_y	updated_weight for next row If y != predicted_y
0.4	0.3	1	0, -0.1, 0.02	0*1 + -0.1*0.4 + 0.02*0.3 = 0 + -0.04 + 0.006 = -0.03	0	0 + 0.1 * 1 = 0.10 -0.1 + 0.1 * 1 * 0.4 = -0.06 0.02 + 0.1 * 1 * 0.3 = 0.05
0.6	0.8	1	0.1, -0.06, 0.05	0.1*1 + -0.06*0.6 + 0.05*0.8 = 0.1 + -0.036 + 0.04 = 0.104	1	
0.7	0.5	1	0.1, -0.06, 0.05	0.1*1 + -0.06*0.7 + 0.05*0.5 = 0.1 + -0.042 + 0.025 = 0.083	1	
0.9	0.2	0	0.1, -0.06, 0.05	0.1*1 + -0.06*0.9 + 0.05*0.2 = 0.1 + -0.054 + 0.01 = 0.056	1	0.1 + 0.1 * -1 = 0.0 -0.06 + 0.1 * -1 * 0.9 = -0.15 0.05 + 0.1 * -1 * 0.2 = 0.03

Epoch = 3						
x1	x2	y	weights	activation units	predicted_y	updated_weight for next row If y != predicted_y
0.4	0.3	1	0, -0.15, 0.03	0*1 + -0.15*0.4 + 0.03*0.3 = 0 + -0.06 + 0.009 = -0.051	0	0 + 0.1 * 1 = 0.10 -0.15 + 0.1 * 1 * 0.4 = -0.11 0.03 + 0.1 * 1 * 0.3 = 0.06
0.6	0.8	1	0.1, -0.11, 0.06	0.1*1 + -0.11*0.6 + 0.06*0.8 = 0.1 + -0.066 + 0.048 = 0.082	1	
0.7	0.5	1	0.1, -0.11, 0.06	0.1*1 + -0.11*0.7 + 0.06*0.5 = 0.1 + -0.077 + 0.03 = 0.053	1	
0.9	0.2	0	0.1, -0.11, 0.06	0.1*1 + -0.11*0.9 + 0.06*0.2 = 0.1 + -0.099 + 0.012 = 0.013	1	0.1 + 0.1 * -1 = 0.0 -0.11 + 0.1 * -1 * 0.9 = -0.2 0.06 + 0.1 * -1 * 0.2 = 0.04
Epoch = 4						
x1	x2	y	weights	activation units	predicted_y	updated_weight for next row If y != predicted_y
0.4	0.3	1	0, -0.2, 0.04	0*1 + -0.2*0.4 + 0.04*0.3 = 0 + -0.08 + 0.012 = -0.068	0	0 + 0.1 * 1 = 0.10 -0.2 + 0.1 * 1 * 0.4 = -0.16 0.04 + 0.1 * 1 * 0.3 = 0.07
0.6	0.8	1	0.1, -0.16, 0.07	0.1*1 + -0.16*0.6 + 0.07*0.8 = 0.1 + -0.096 + 0.056 = 0.06	1	
0.7	0.5	1	0.1, -0.16, 0.07	0.1*1 + -0.16*0.7 + 0.07*0.5 = 0.1 + -0.112 + 0.035 = 0.023	1	
0.9	0.2	0	0.1, -0.16, 0.07	0.1*1 + -0.16*0.9 + 0.07*0.2 = 0.1 + -0.144 + 0.014 = -0.03	0	

Epoch = 5						
x1	x2	y	weights	activation units	predicted_y	updated_weight for next row If y != predicted_y
0.4	0.3	1	0.1, -0.16, 0.07	0.1*1 + -0.16*0.4 + 0.07*0.3 = 0.1 + -0.064 + 0.021 = 0.057	1	
0.6	0.8	1	0.1, -0.16, 0.07	0.1*1 + -0.16*0.6 + 0.07*0.8 = 0.1 + -0.096 + 0.056 = 0.06	1	
0.7	0.5	1	0.1, -0.16, 0.07	0.1*1 + -0.16*0.7 + 0.07*0.5 = 0.1 + -0.112 + 0.035 = 0.023	1	
0.9	0.2	0	0.1, -0.16, 0.07	0.1*1 + -0.16*0.9 + 0.07*0.2 = 0.1 + -0.144 + 0.014 = -0.03	0	

6-வது *epoch*-ல் தான், அனைத்துப் பயிற்சித் தரவுகளும் சரியாக கணிக்கப்படுகின்றன. எனவே அதனுடைய *weights*-ஐயே பிற்காலத் தரவுகளை கணிப்பதற்கான *algorithm*-ன் *weights*-ஆக நாம் எடுத்துக் கொள்ளலாம். இவ்வாறாக முதல் தரவின் கணிப்பு சரியாக இருந்தால், அடுத்த தரவிற்குச் செல்லும். இல்லையெனில் *weights*-ஐ மீண்டும் கணக்கிட்டு அடுத்த தரவிற்குச் செல்லும். கணிப்புகள் அனைத்தும் சரியாக நிகழும் வரை இதே முறை பின்பற்றப்படுவதால், இது *error-driven learning algorithm* என்று அழைக்கப்படுகிறது. இதனடிப்படையில் அமைகின்ற *MLP (Multiple Linear Perceptron)* என்பதே *neural networks*-ஐ உருவாக்குகிறது.

Artificial Neural Networks

ஒரு நியூரான் கற்றுக் கொள்வதை அடிப்படையாக வைத்து கற்றுக் கொள்வது *perceptron* என்றால், பல்வேறு நியூரான்களைக் கொண்ட மனித மூளை கற்றுக் கொள்வதை அடிப்படையாக வைத்து கற்றுக் கொள்வது *Multi-layer perceptron* ஆகும். அதாவது செயல்களை அடிப்படையாகக் கொண்டு நியூரான்கள் கற்கின்றன. நியூரான்கள் கற்றுக் கொண்டதை வைத்து மனித மூளை கற்கிறது. இதே முறையில் தரவுகளை அடிப்படையாகக் கொண்டு *perceptron* கற்கின்றன. *Perceptron*-களை வைத்து *directed acyclic graph*-ஐ உருவாக்கி *MLP* கற்கிறது. இதுவே *Artificial neural network* என்று அழைக்கப்படுகிறது.

Perceptron என்பது நேர்கோடு மூலம் பிரிக்கக்கூடிய தரவுகளை வகைப்படுத்த உதவும் என்று ஏற்கெனவே பார்த்தோம். *Non-linear* முறையில் அமைந்துள்ள தரவுகளைப் பிரிப்பதற்கு *MLP*-ஐப் பயன்படுத்தலாம். எனவேதான் இது *universal function approximator* என்று அழைக்கப்படுகிறது. இது தவிர *kernalization* என்ற தத்துவமும் *non-linear* முறையில் அமைந்துள்ள தரவுகளைப் பிரிப்பதற்கு உதவும். இதைப்பற்றி *SVM* என்ற பகுதியில் ஏற்கெனவே பார்த்து விட்டோம்.

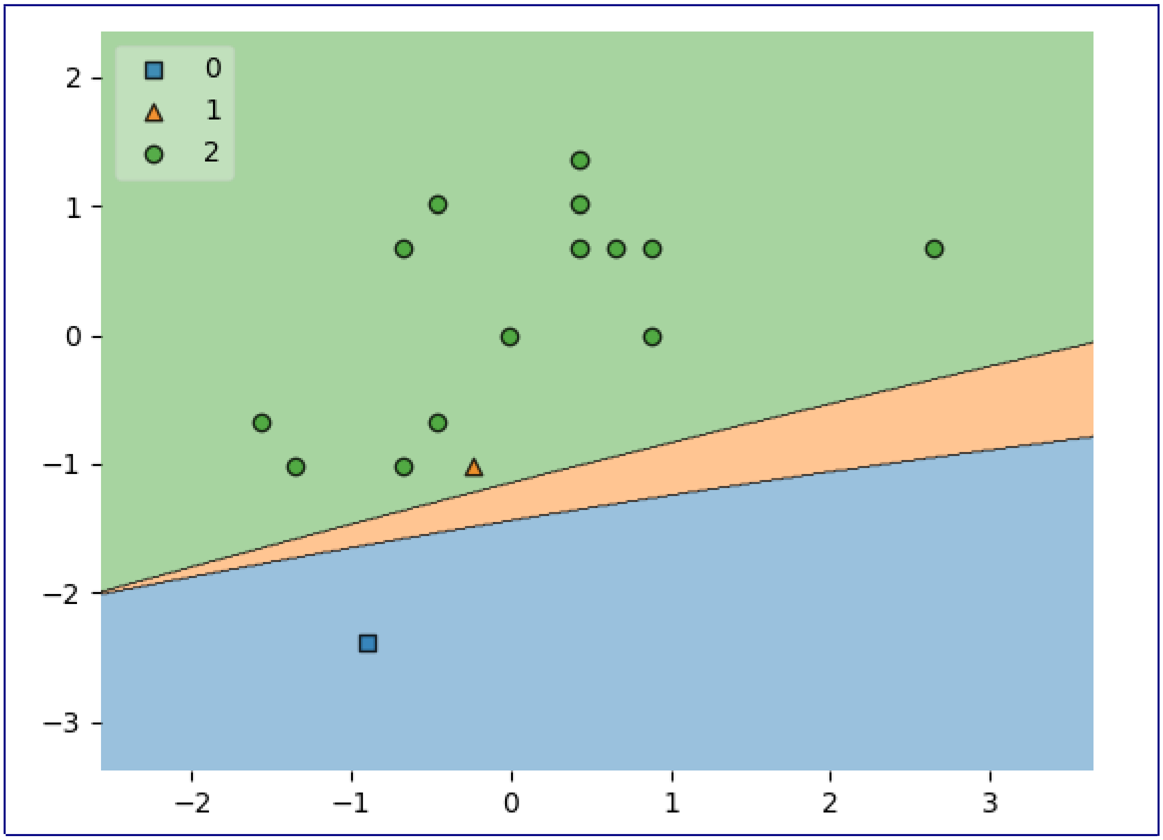
கீழ்க்கண்ட எடுத்துக்காட்டில், *16* பயிற்சித் தரவுகள் கொடுக்கப்பட்டுள்ளன. *X* -ல் அதனுடைய *2 features*-ம், *y*-ல் அவை எந்த வகையின் கீழ் பிரிக்க வேண்டும் எனும் விவரமும் பயிற்சிக்கு அளிக்கப்பட்டுள்ளன. *1,2,3* எனும் மூன்று வகைகளின் கீழ் தரவுகள் பிரிக்கப்படுவதால் இது *multi-class classification*-க்கான உதாரணம் ஆகும்.

https://gist.github.com/nithyadurai87/b95e0ccd56464646da32ffdddb8b457f

```
from mlxtend.classifier import MultiLayerPerceptron as MLP
from mlxtend.plotting import plot_decision_regions
import matplotlib.pyplot as plt
import numpy as np

X = np.asarray([[6.1,1.4],[7.7,2.3],[6.3,2.4],[6.4,1.8],[6.2,1.8],[6.9,2.1],
[6.7,2.4],[6.9,2.3],[5.8,1.9],[6.8,2.3],[6.7,2.5],[6.7,2.3],[6.3,1.9],[6.5,2.1 ],
[6.2,2.3],[5.9,1.8]] )

X = (X - X.mean(axis=0)) / X.std(axis=0)

y = np.asarray([0,2,2,1,2,2,2,2,2,2,2,2,2,2,2,2])

nn = MLP(hidden_layers=[50],l2=0.00,l1=0.0,epochs=150,eta=0.05,
momentum=0.1,decrease_const=0.0,minibatches=1,random_seed=1,print_progress=3)
nn = nn.fit(X, y)

fig = plot_decision_regions(X=X, y=y, clf=nn, legend=2)
plt.show()
print('Accuracy(epochs = 150): %.2f%%' % (100 * nn.score(X, y)))

nn.epochs = 250
nn = nn.fit(X, y)
fig = plot_decision_regions(X=X, y=y, clf=nn, legend=2)
plt.title('epochs = 250')
plt.show()
print('Accuracy(epochs = 250): %.2f%%' % (100 * nn.score(X, y)))

plt.plot(range(len(nn.cost_)), nn.cost_)
plt.title('Gradient Descent training (minibatches=1)')
plt.xlabel('Epochs')
plt.ylabel('Cost')
plt.show()

nn.minibatches = len(y)
```

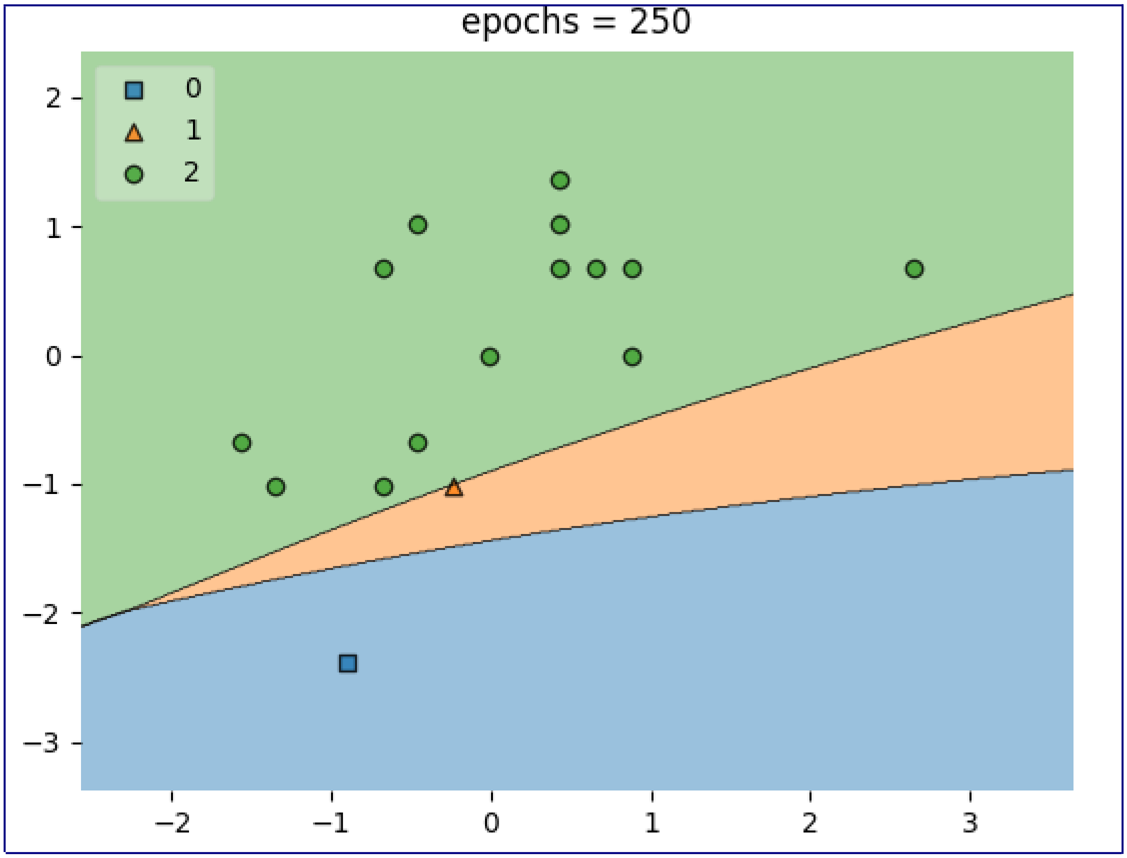
```
nn = nn.fit(X, y)
plt.plot(range(len(nn.cost_)), nn.cost_)
plt.title('Stochastic Gradient Descent (minibatches=no. of training examples)')
plt.ylabel('Cost')
plt.xlabel('Epochs')
plt.show()
```

பயிற்சித் தரவுகளைக் கொண்டு MLP-க்குப் பயிற்சி அளிக்கும்போது, அது பின்வருமாறு தரவுகளைப் பிரிக்கிறது. இதில் 1 எனும் வகையின் கீழ் பிரிக்கப்பட வேண்டியது அதற்குரிய இடத்தில் சரியாக அமையாமல், 0 எனும் வகையின்கீழ் பிரிக்கப்பட்டிருப்பதைக் காணலாம்.



எனவே MLP-க்குப் பயிற்சி அளிக்கும்போது கொடுக்கப்பட்டுள்ள epochs-ன் எண்ணிக்கையை 150-லிருந்து 250-என மாற்றி பயிற்சி அளித்துப் பார்க்கவும். இப்போது அனைத்துத் தரவுகளும் சரியாக வகைப்படுத்தப்பட்டிருப்பதைக் காணலாம்.

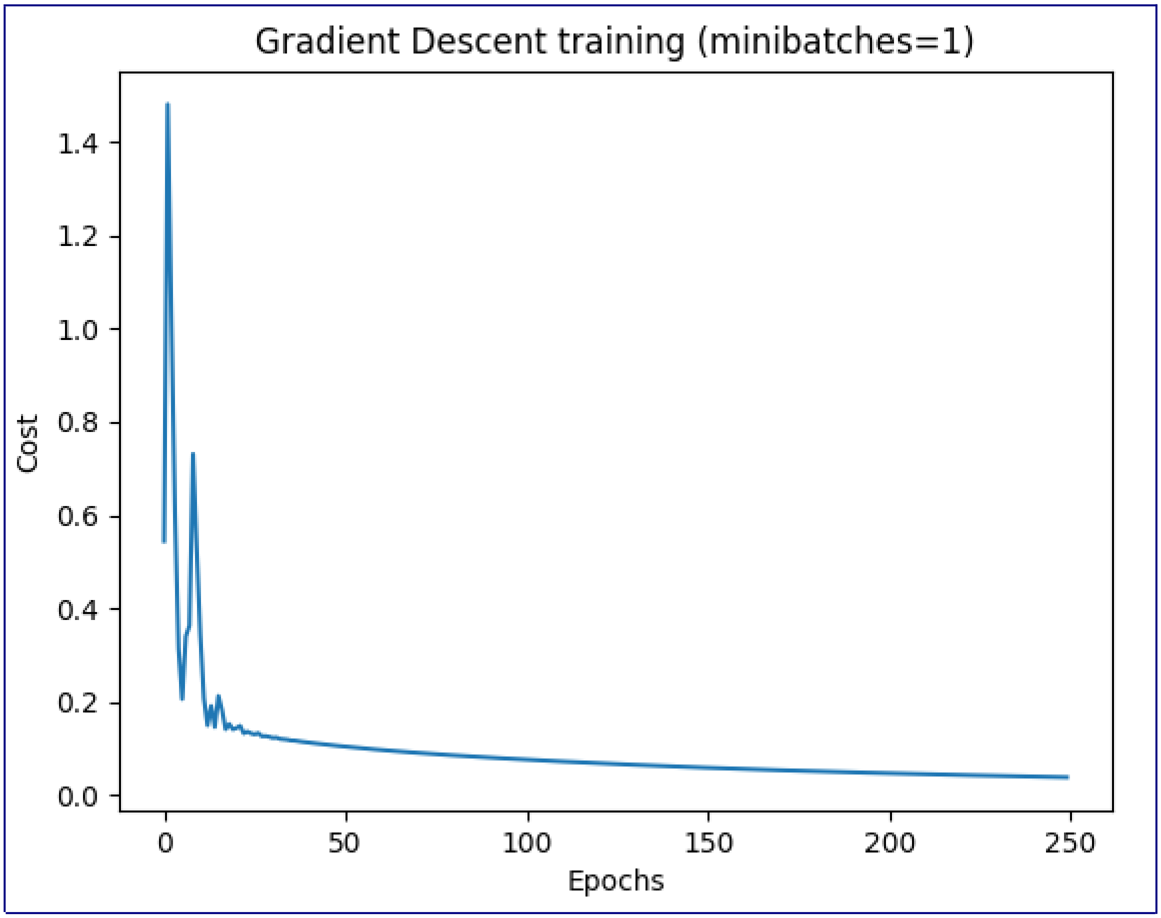
```
nn.epochs = 250
nn = nn.fit(X, y)
fig = plot_decision_regions(X=X, y=y, clf=nn, legend=2)
plt.title('epochs = 250')
plt.show()
```



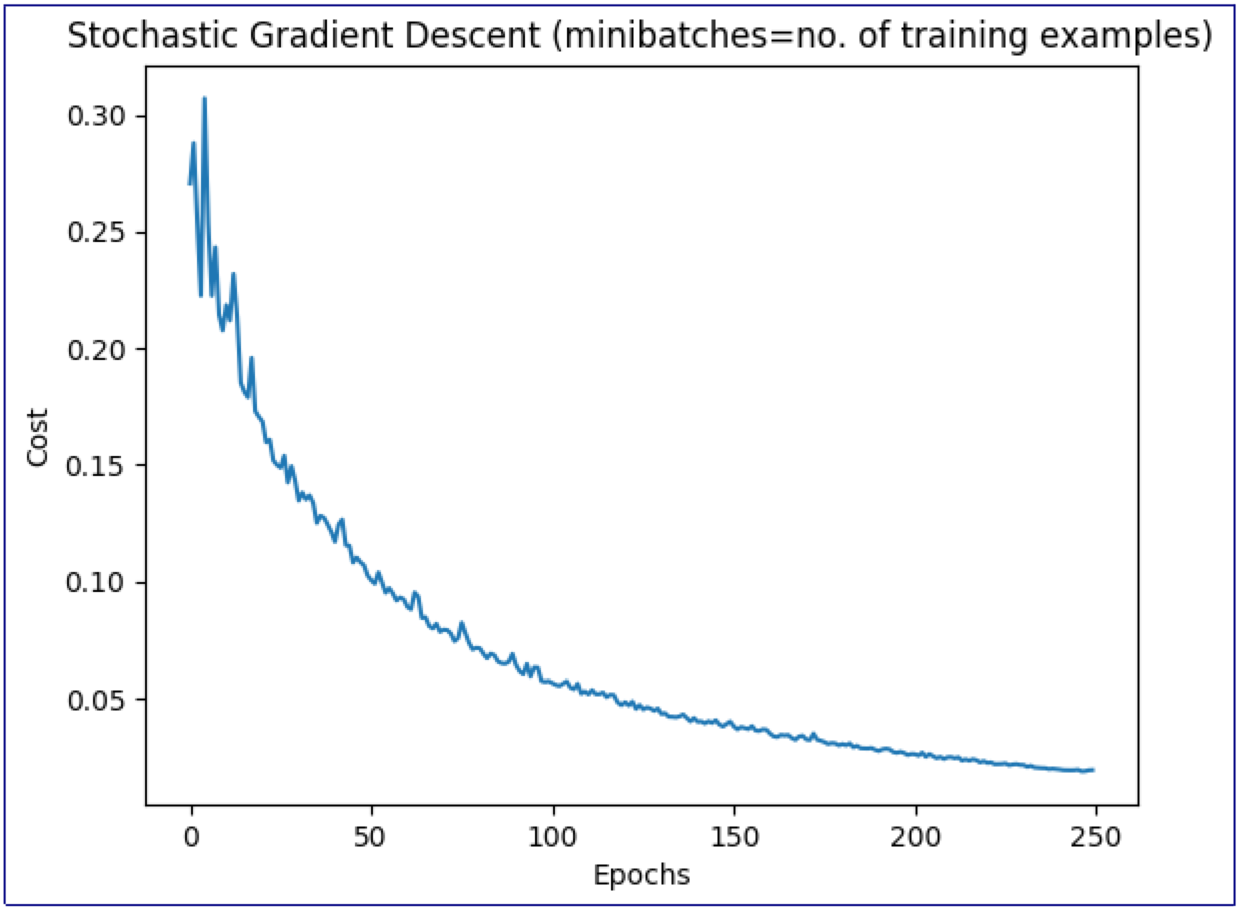
எனவேதான் 150 எனும்போது அதன் accuracy 93.75% எனவும், 250 எனும்போது அதன் accuracy 100.00% எனவும் உயர்ந்திருப்பதைக் காணலாம்.

Iteration: 150/150 | Cost 0.06 | Elapsed: 0:00:00 | ETA: 0:00:00Accuracy(epochs = 150): 93.75%
Iteration: 250/250 | Cost 0.04 | Elapsed: 0:00:00 | ETA: 0:00:00Accuracy(epochs = 250): 100.00%

அடுத்ததாக ஒவ்வொரு epoch-லும் அதன் cost மதிப்பு எவ்வாறு குறைகிறது என்பது வரைபடமாக வரைந்து காட்டப்படுகிறது. MLP-க்குப் பயிற்சி அளிக்கும்போது கொடுக்கப்பட்டுள்ள parameter-களில் ஒன்றான minibatches -ன் மதிப்பு 1 என அமைந்தால் gradient descent முறையில் தரவுகளுக்குப் பயிற்சி அளிக்கும். இதைப் பற்றித்தான் perceptron-ல் கற்றோம்.



அதுவே *minibatches* -ன் மதிப்பு கொடுக்கப்பட்டுள்ள மாதிரித் தரவுகளின் எண்ணிக்கையாக அமைந்தால், அது *stochastic gradient descent* முறையில் தரவுகளுக்குப் பயிற்சி அளிக்கும். அதாவது ஒவ்வொரு தரவுகளாக சோதித்து *gradient* முறையில் *cost* மதிப்பை குறைத்து வராமல், மொத்தமாக அனைத்துப் பயிற்சித் தரவுகளையும் எடுத்துக் கொண்டு குறைந்த *cost*-ஐக் கண்டுபிடிக்க உதவுவதே *stochastic gradient descent* ஆகும். எனவேதான் கீழ்க்கண்ட வரைப்படத்தில், ஒவ்வொரு *epoch*-லும் அதனுடைய *cost* மதிப்பு அனைத்துப் பயிற்சித் தரவுகளையும் சேர்த்து கணக்கிடப்படுவதால், அவை *zig-zag* வடிவில் அமைந்திருப்பதைக் காணலாம். மிக அதிக அளவு எண்ணிக்கையில் பயிற்சித் தரவுகள் இருக்கும்போது, *gradient descent* அவை அனைத்தையும் ஒவ்வொன்றாக ஆராய்ந்து *global optimum* சென்றடைய மிகுந்த நேரம் பிடிக்கும். ஆகையால், இது போன்ற தருணங்களில் *stochastic*-ஐப் பயன்படுத்தலாம். இதுவே *batch gradient descent* என்றும் அழைக்கப்படுகிறது.


Stochastic Gradient Descent (minibatches=no. of training examples)
Cost
0.30
0.25
0.20
0.15
0.10
0.05
0
50
100
150
200
250
Epochs