BA713 - Machine Learning & AI

END-TO-END MACHINE LEARNING PROJECT

(Module 2)

CONTENTS

- ML/DL Pipeline
- Data Pre-processing
- Feature selection
- Normalization versus Standardization
- Dimension Reduction
- Training and evaluation
- Evaluation Metrics
- Hyperparameter optimization
- Building an end-to-end ML project

TEXTBOOKS REFERRED

1. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems by Aurelien Geron, first edition Published by O’Reilly Media, Inc. 2017, ISBN: 978-1-491-96229-9

2. Deep Learning with Python by François Chollet, published by Manning Publications Co., 2018, ISBN: 978-1-61729-443-3

ML/DL PIPELINE

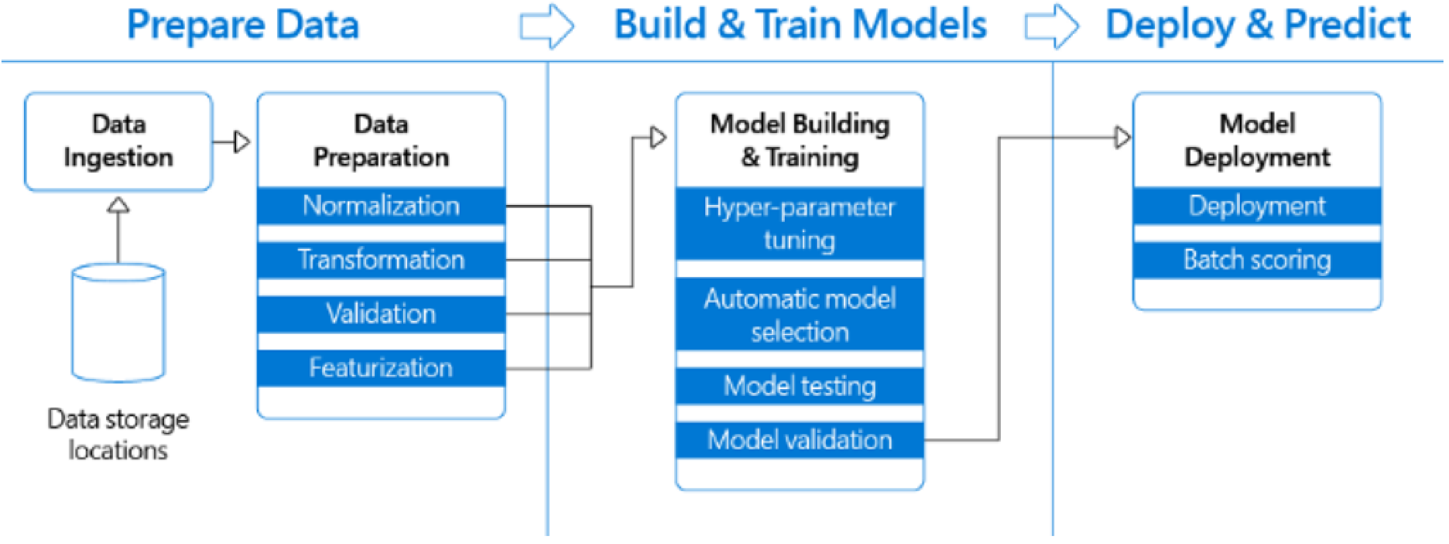


Image Source: https://medium.com/microsoftazure/how-to-accelerate-devops-with-machine-learning-lifecycle-management-2ca4c86387a0

DATA PRE-PROCESSING IN ML

- Crucial step
- Select, Transform, Augment data
- Feature engineering→ domain knowledge + data science

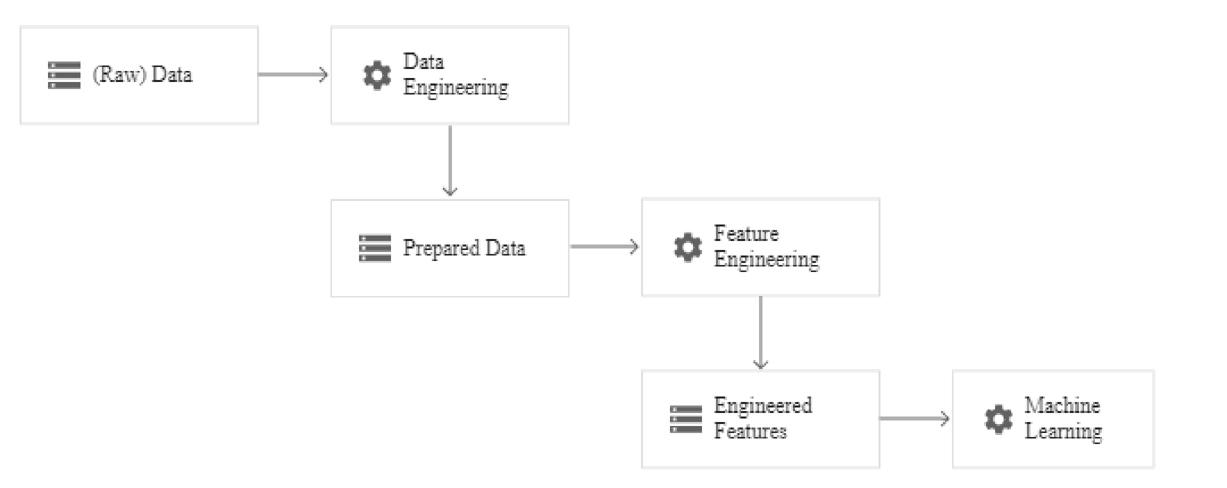
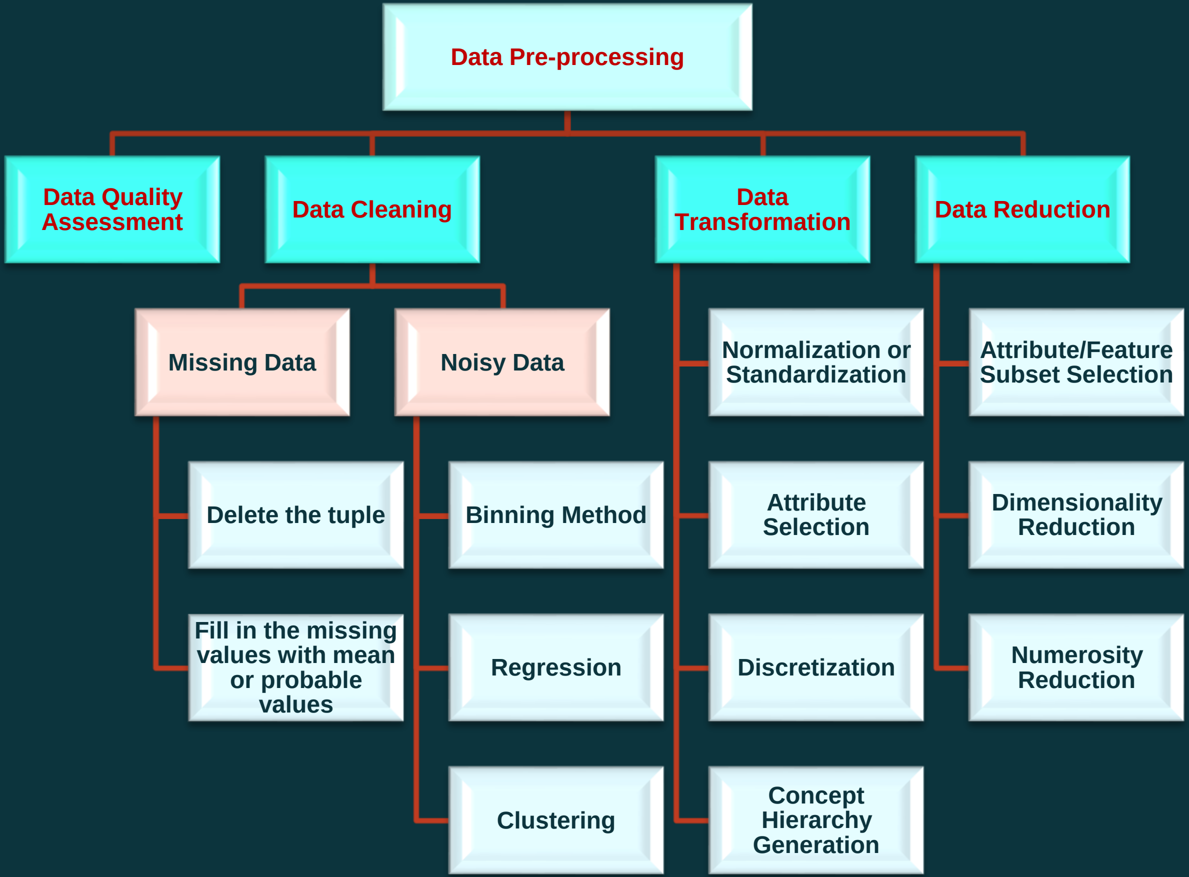


Figure: Data Pre-processing for ML

Source: https://cloud.google.com/architecture/data-preprocessing-for-ml-with-tf-transform-pt1

DATA PRE-PROCESSING CATEGORIZATION



DATA PRE-PROCESSING IN ML

1. Data Quality Assessment:

irrelevant bits, mismatching data types (float, int), different dimensions of data arrays, mixture of data values (man, male), outliers (daily temperature high, few days very low), missing data

Figure: Noisy Data

Need for pre-processing:

- Accuracy of data→ eliminate missing values
- Reduce inconsistencies→ duplicates, noise
- Complete Information→ missing attributes
- Smoothen the data→ Easy to use and interpret

Source: https://serokell.io/blog/data-preprocessingc

DATA PRE-PROCESSING IN ML

2. Data Cleaning:

- Missing data/ values→ collect more data from other sources, add missing data
- Noisy data→ meaningless information in data→ noise, duplicates, unnecessary information
- For example, Identify if a person speaks English or not, feature provided is shoe size, blood pressure, etc.

For Noisy Data:

a) *Binning*→ Pool of sorted data, For example, age binned into categories 16-21, 22-35, 36-45, 45-60, etc.

b) *Regression Analysis*→ Identify relationship between features and label, identify key features making an impact

c) *Clustering*→ For example, K-Means→ group data→ identify outliers

Figure: Outliers in Data

Source: https://serokell.io/blog/data-preprocessing

DATA PRE-PROCESSING IN ML

3. Data Transformation or Scaling:

- Tuning of data in a proper format
- For example, data about car speed in different formats such as miles per second or kilometers per second

Categories:

a) *Aggregation*→ Data from different sources aggregated in a single format, e.g., image data (jpeg, png)

b) *Normalization/ Standardization*→ Scaling of data within a range, e.g., Population size

c) *Feature or attribute selection*→ Reduce unnecessary features, improve accuracy

d) *Discretization*→ improve efficiency, e.g., creating categories for age "young", "middle-aged" and "senior"

e) *Concept Hierarchy Generation*→ hierarchical relationships between features, e.g., address includes street, city, province , country

f) *Generalization*→ low-level to high-level e.g., house address to town and country

Source: https://serokell.io/blog/data-preprocessing

DATA PRE-PROCESSING IN ML

4. Data Reduction:

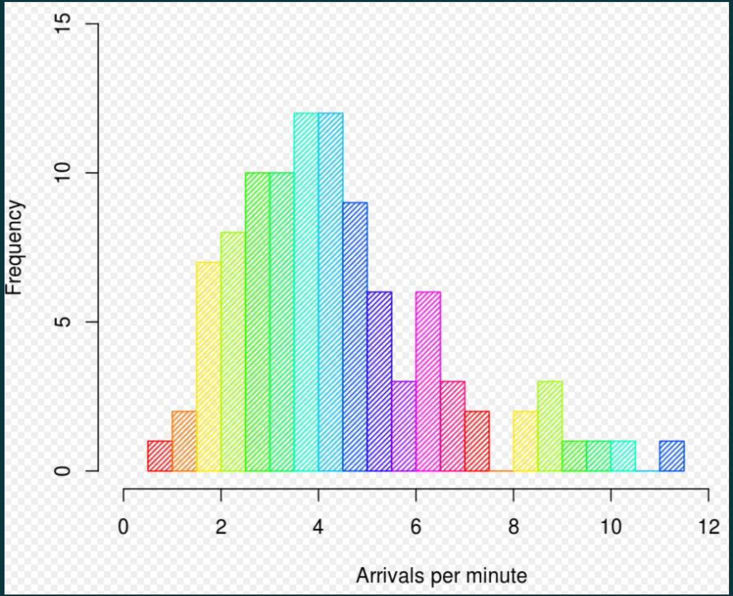
- Reduce amount of data/features, decrease the analysis cost

Categories:

a) *Attribute/Feature Subset Selection*→ For example, Irrelevant features e.g., IP address prediction of a network attack, Redundant features e.g., purchase price of a product and sales tax paid

b) *Dimensionality reduction*→ Computer Vision, Speech Generation→ speakers filling words not important→ can be sampled

c) *Numerosity reduction*→ replace original data by a smaller form of data representation, data sampling, histogram, identify patterns in data



Source: https://en.wikipedia.org/wiki/Histogram

Source: https://serokell.io/blog/data-preprocessing

POPULAR OPEN DATA SOURCES

- Popular open data repositories

—UC Irvine Machine Learning Repository (UCI)
—Kaggle datasets
—Amazon’s AWS datasets

- Meta portals (they list open data repositories)

—Data Portals
—OpenDataMonitor
—Quandl

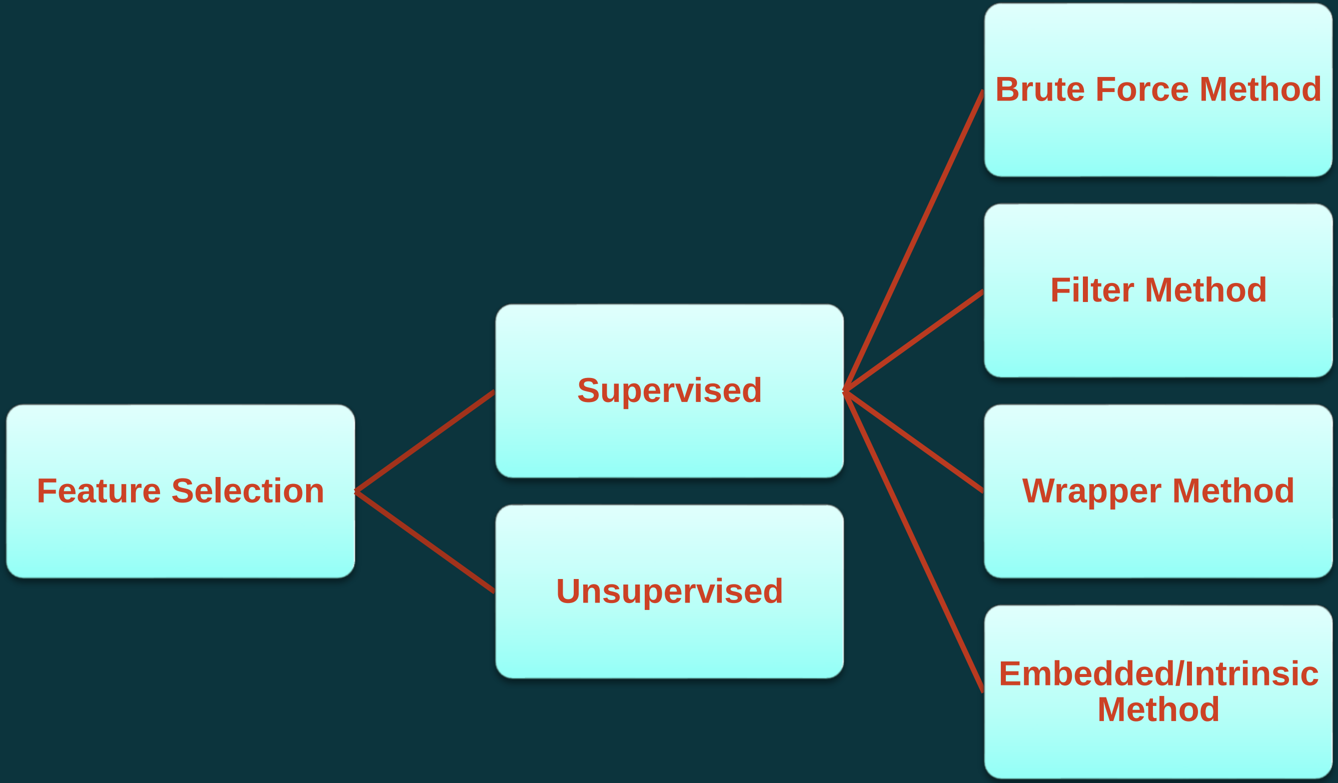
- Other pages listing many popular open data repositories

—Wikipedia’s list of Machine Learning datasets
—Quora.com
—The datasets subreddit

FEATURE SELECTION TECHNIQUES

Why Feature Selection?

- Faster training
- Reduced complexity
- Improves Accuracy
- Reduces overfitting


Feature Selection

Supervised

Unsupervised

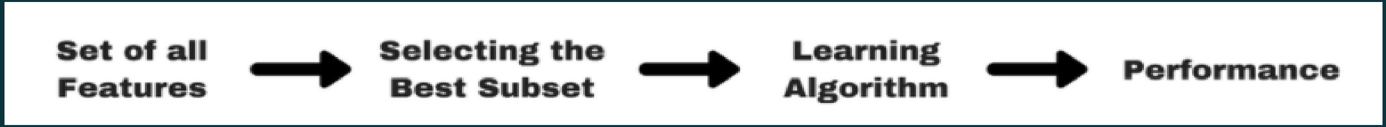
Brute Force Method

Filter Method

Wrapper Method

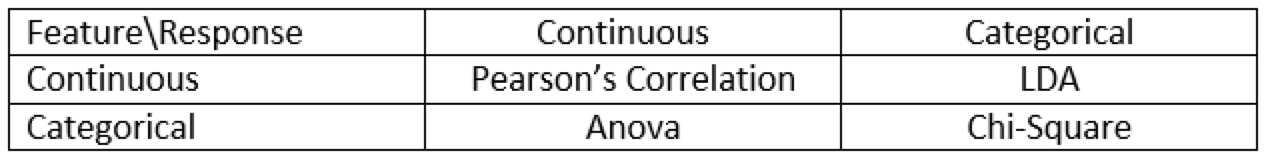
Embedded/Intrinsic Method

FILTER METHOD FOR FEATURE SELECTION



Set of all Features → Selecting the Best Subset → Learning Algorithm → Performance

- Pre-processing step
- Feature selection→ basis of scores in statistical tests→ correlation with target variable



Feature\Response	Continuous	Categorical
Continuous	Pearson’s Correlation	LDA
Categorical	Anova	Chi-Square

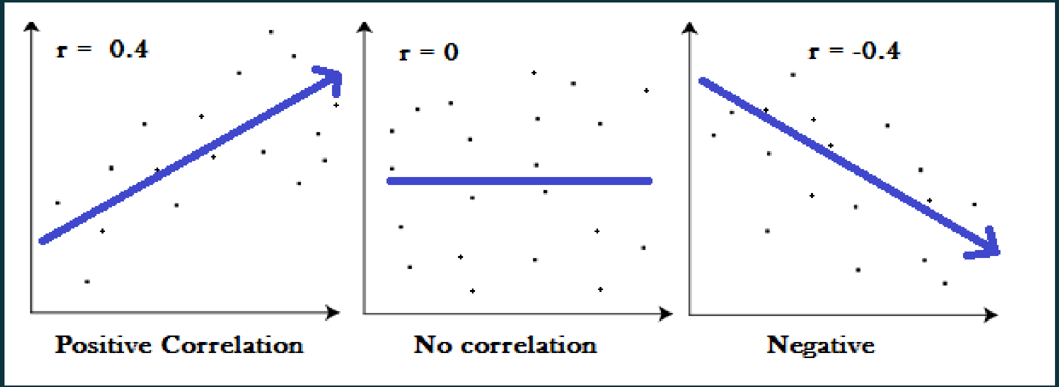
- *Pearson’s Correlation:* linear dependence between two continuous variables X and Y. Value ranges from -1 to +1
- *Linear Discriminant Analysis (LDA):* linear combination of features that separates two or more classes
- *Analysis of Variance (ANOVA):* statistical test to compare if the means of several groups of data belong to same distribution
- *Chi-Square:* statistical measure of correlation between a categorical feature and a categorical target value

Source: https://www.analyticsvidhya.com

PEARSON'S CORRELATION

- Correlation→ strength of a relationship between two variables
- Pearson's correlation→ linear regression
- Returns values between -1 and +1

$$r = \frac{n(\Sigma xy) - (\Sigma x)(\Sigma y)}{\sqrt{[\, n\Sigma x^2 - (\Sigma x)^2]\,[\, n\Sigma y^2 - (\Sigma y)^2]}}$$



Figures showing a correlation of -1, 0 and +1

- +1→perfect positive correlation
- -1→perfect negative correlation
- 0→no correlation
- Between +/- 0.5 and +/- 1→ strong correlation
- Between +/- 0.3 and +/-0.49→ medium correlation
- Below +/- 0.29→ small correlation

Source: https://www.statisticshowto.com/

CHI-SQUARE

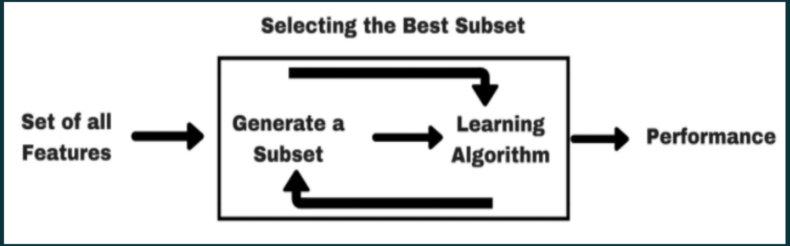
- Compares two variables to check if they are related
- In the equation, the subscript “c” is the degrees of freedom. “O” is your observed value and “E” is your expected value.
- relationship between two categorical variables
- Low chi-square value→ little relationship between the two variables
- In theory, if O=E, Chi-square=0→ unlikely in real life
- Compare chi-square statistic with a critical value in Chi-Square distribution table

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

Source: https://www.statisticshowto.com/

WRAPPER METHOD FOR FEATURE SELECTION

- Employ a feature subset to train the ML model
- Based on model results→ search the best features for performance improvement
- Computationally expensive



- *Forward Selection:*

✓ start no feature, each iteration add features that best improve model results until not improve

- *Backward Elimination:*

✓ start all features, remove least significant features each iteration until no improvement

- *Recursive Feature Elimination:*

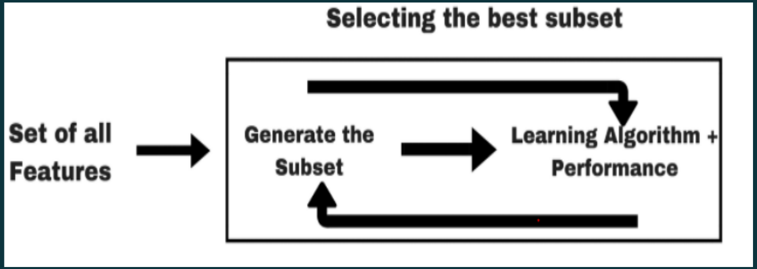
✓ greedy optimization, each iteration best and worst feature set aside

✓ Next model→ features left until all exhausted, Rank features based on elimination order

Source: https://www.analyticsvidhya.com

EMBEDDED/INTRINSIC METHOD FOR FEATURE SELECTION

- Combine the characteristics of both filter as well as wrapper methods
- ML algorithms with built-in feature selection
- For example, regularization or penalization methods
- Add constraints to optimization of an AI algorithm (for example, regression algorithm)



Examples:

a) *Lasso Regression*→ Linear Regression, L1 regularization, shrink data values at the mean, reduce # features

b) *Ridge Regression*→ L2 regularization, reducing model complexity and overfitting

c) *Tree based algorithms* such as decision tree, random forest, etc. also support intrinsic feature selection functions

Source: https://www.analyticsvidhya.com

LASSO REGRESSION- L1 REGULARIZATION

- Least Absolute Shrinkage and Selection Operator or LASSO
- Shrink data values at a central point or mean
- Sparse models→ few parameters
- Reduce overfitting
- When multiple features are highly correlated→ multicollinearity
- L1 Regularization adds penalty equal to the absolute value of the magnitude of coefficients.
- Larger penalties result in coefficient values closer to zero thus eliminating certain features

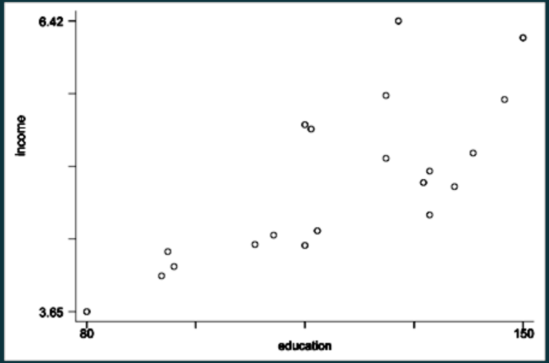


Figure: Multicollinearity affects Regression adversely

- Lasso Regression with L1 Regularization aims to minimize the following equation:

$$L_{lasso} = argmin_{\hat{\beta}} \left(\|Y - \beta * X\|^2 + \lambda * \|\beta\|_1 \right)$$

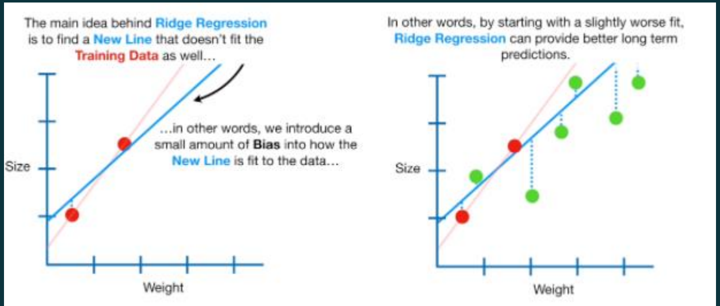
Source: https://www.statisticshowto.com/

RIDGE REGRESSION- L2 REGULARIZATION

Ridge Regression with L2 Regularization aims to minimize the following equation:

$$L_{ridge} = argmin_{\hat{\beta}} \left(\|Y - \beta * X\|^2 + \lambda * \|\beta\|_2^2 \right)$$

- Shrinkage/Ridge estimator λ→ L2 Penalty →square of the magnitude of coefficients
- Reducing model complexing and overfitting
- When Tuning parameter λ = 0, penalty=0, minimize sum of squared residuals
- If λ = ∞, all coefficients are shrunk to zero
- Ideal penalty → between 0 and ∞



The main idea behind Ridge Regression is to find a New Line that doesn't fit the Training Data as well...

...in other words, we introduce a small amount of Bias into how the New Line is fit to the data...

In other words, by starting with a slightly worse fit, Ridge Regression can provide better long term predictions.

https://www.andreaperlato.com/theorypost/ridge-and-lasso-regression/

LASSO VERSUS RIDGE REGRESSION

Penalty Function:

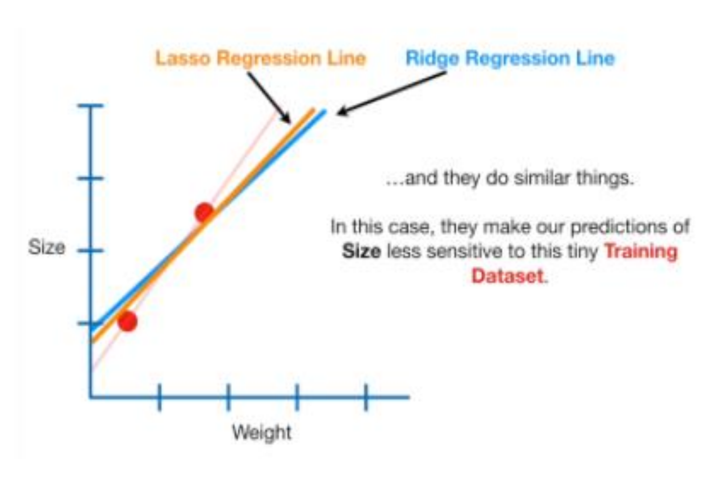
- Lasso→ L1 Penalty function: λ |slope| → important features
- Ridge→L2 Penalty function : λ *slope^2 → none of the coefficients are extremely large

Slope:

- Lasso→ shrink the slope to zero
- Ridge→ shrink the slope close to zero

Features:

- Lasso→ Increase λ, most important features shrink a bit but less important are zero and thus can be eliminated
- Ridge→ When λ is increased, most important features shrink little bit, but less important parameters are not minimized to zero



...and they do similar things.

In this case, they make our predictions of **Size** less sensitive to this tiny **Training Dataset**.

https://www.andreaperlato.com/theorypost/ridge-and-lasso-regression/

FEATURE SCALING FOR MACHINE LEARNING

- Dataset with different magnitude, range or units for features
- Impediment for some ML algorithms

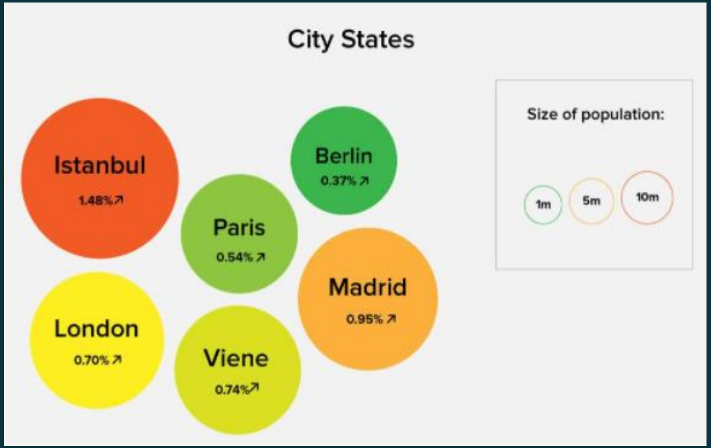


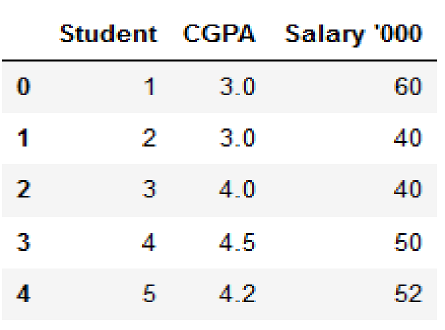
Image Source: https://serokell.io/blog/data-preprocessing

Why do we need scaling?

- Different ranges of values for population→ some algorithms sensitive to such information
- Gradient Descent Based Algorithms such as Linear Regression, Logistic Regression and Neural Networks
- Distance algorithms like KNN, K-means, and SVM→ range of features

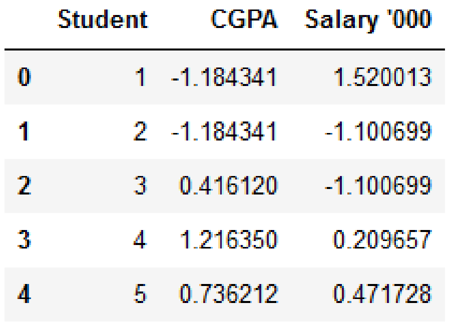
Source: https://www.analyticsvidhya.com

EFFECT OF FEATURE SCALING EXAMPLE



	Student	CGPA	Salary '000
0	1	3.0	60
1	2	3.0	40
2	3	4.0	40
3	4	4.5	50
4	5	4.2	52

Table: Values of Features Before Scaling



	Student	CGPA	Salary '000
0	1	-1.184341	1.520013
1	2	-1.184341	-1.100699
2	3	0.416120	-1.100699
3	4	1.216350	0.209657
4	5	0.736212	0.471728

Table: Values of Features After Scaling

Compare Euclidean Distance between data points for students 0 & 1 and between 1 & 2

Distance between 0 & 1 before scaling $\sqrt{(40-60)^2+(3-3)^2}=20$

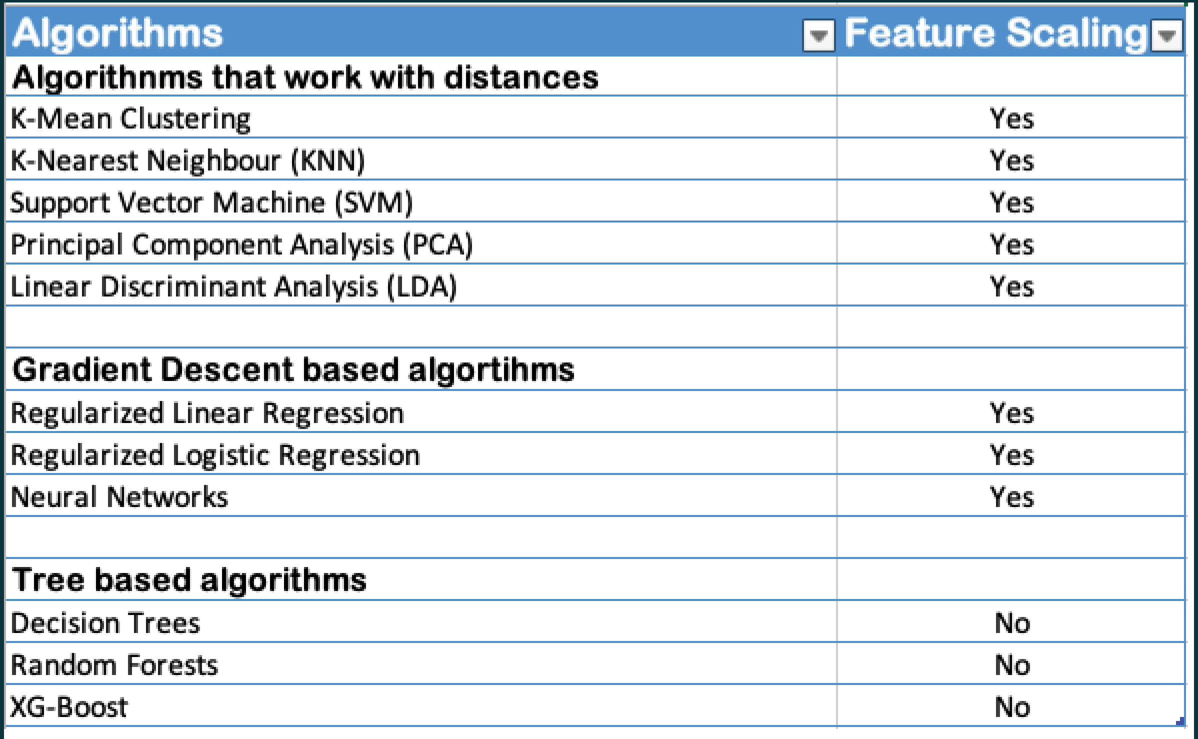
Distance between 1 & 2 before scaling $\sqrt{(40-40)^2+(4-3)^2}=1$

Distance between 0 & 1 after scaling $\sqrt{(1.1+1.5)^2+(1.18-1.18)^2}=2.6$

Distance between 1 & 2 after scaling $\sqrt{(1.1-1.1)^2+(0.41+1.18)^2}=1.59$

Source: https://www.analyticsvidhya.com

ALGORITHMS THAT NEED FEATURE SCALING



Algorithms	Feature Scaling
Algorithnms that work with distances	
K-Mean Clustering	Yes
K-Nearest Neighbour (KNN)	Yes
Support Vector Machine (SVM)	Yes
Principal Component Analysis (PCA)	Yes
Linear Discriminant Analysis (LDA)	Yes
Gradient Descent based algortihms	
Regularized Linear Regression	Yes
Regularized Logistic Regression	Yes
Neural Networks	Yes
Tree based algorithms	
Decision Trees	No
Random Forests	No
XG-Boost	No

Source: https://becominghuman.ai/feature-scaling-in-machine-learning-20dd93bb1bcb

NORMALIZATION/ MIN-MAX SCALING

- Values are scaled in the range between 0 and 1. It is also known as Min-Max scaling.
- The formula is given below:

$$X' = \frac{X - Xmin}{Xmax - Xmin}$$

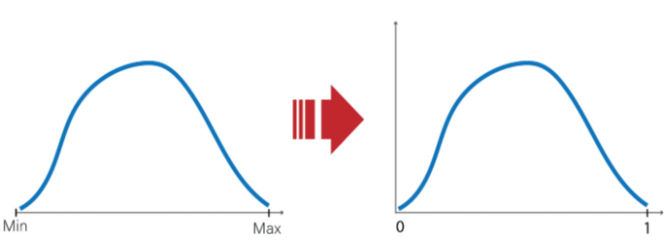


Figure: Normalization (Min-Max Scaling)

Here, Xmax and Xmin are the maximum and the minimum values of a feature respectively.

- When the value of X (feature) is the minimum→ numerator will be 0, and hence X' = 0
- When the value of X is the maximum→ numerator = denominator, and X' = 1
- If the value of X is between the minimum and the maximum value→ value of X' is between 0 and 1

Source: https://www.analyticsvidhya.com

STANDARDIZATION (Z-SCORE)

- Values are scaled with a mean of zero and standard deviation of one
- mean of the attribute becomes zero and the resultant distribution has a unit standard deviation.
- The formula is given below:

$$z = \frac{x - \mu}{\sigma}$$

- Here, "μ" is the mean of the feature values
- "σ " is the standard deviation of the feature values

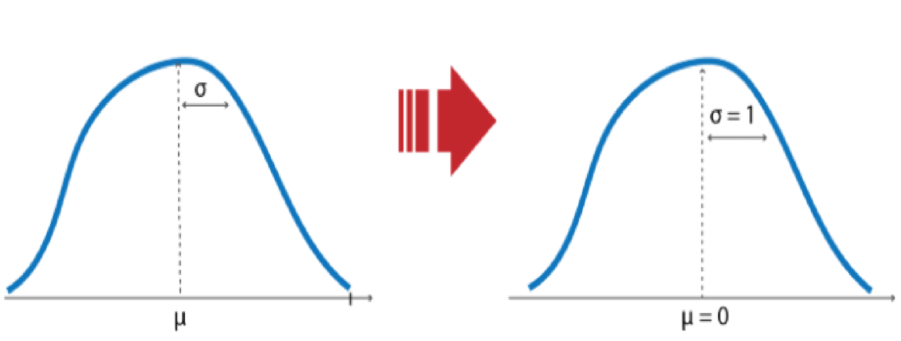


Figure: Standardization (Z-Score)

Source: https://www.analyticsvidhya.com

NORMALIZATION VERSUS STANDARDIZATION

Data Distribution:

- Standardization→ the data follows a Gaussian distribution. However, this does not have to be necessarily true
- Normalization→ No Gaussian distribution, useful in algorithms that do not assume any data distribution such as K-Nearest Neighbors and Neural Networks

Outliers Handling:

- Standardization→ more robust to outliers, does not have a bounding range, facilitates faster convergence of loss function for some algorithms
- Normalization→ transform all the features with varying scales to a common scale but does not handle outliers well

Robustness to new data:

- Standardization is more robust to new data since it applies mean and standard deviation
- Normalization is less robust to new data since it employs min and max values

Source: https://www.analyticsvidhya.com

CURSE OF DIMENSIONALITY

- ML problems→ huge datasets with thousands & millions of features
- Slow convergence time
- Not an optimal solution

Solution to this problem:

- Reducing number of features considerably→ dimension reduction
- For example, MNIST Dataset, drop pixel information from image borders→ always white and irrelevant (see figure)
- Merging two highly correlated pixels into one

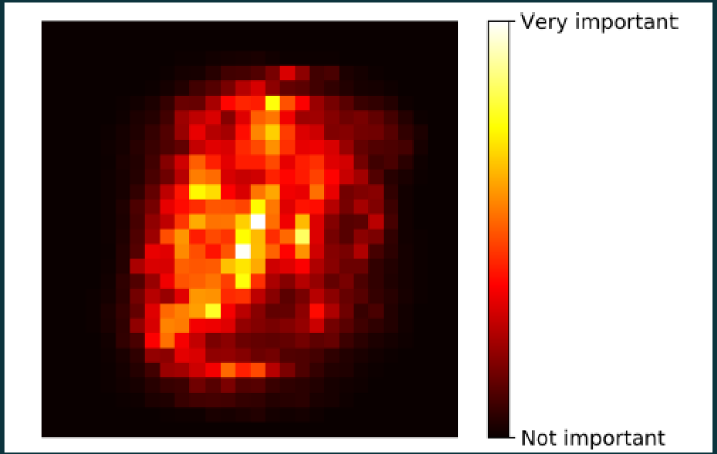


Figure: MNIST pixel importance (according to a Random Forest classifier)

Source: Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow

PRINCIPAL COMPONENT ANALYSIS (PCA)

- Most popular→ dimension reduction
- Identify the hyperplane that lies closest to the data, and then it projects the data onto it
- Choose the right hyperplane
- Preserve variance→ lose less information
- Choose the axis that minimizes the mean squared distance between the original dataset and its projection onto that axis

c1 and c2 are the first & second principal components

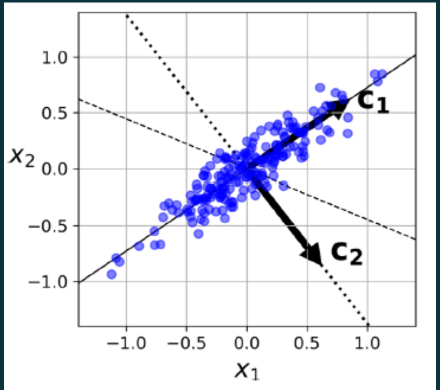
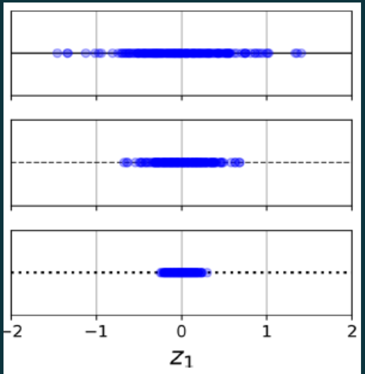


Figure (a): 2D dataset with 3 axes (1D hyperplanes)



Preserves maximum variance

Preserves intermediate amount of variance

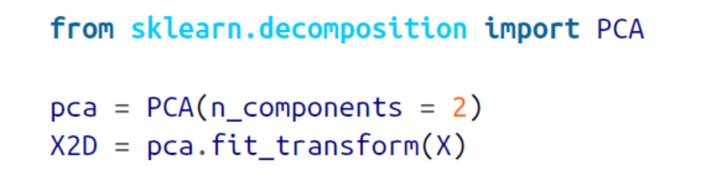
Preserves very little variance

Figure (b): Selecting the subspace to project on

Source: Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow

PCA USING SCIKIT-LEARN

- uses singular value decomposition (SVD) to implement PCA

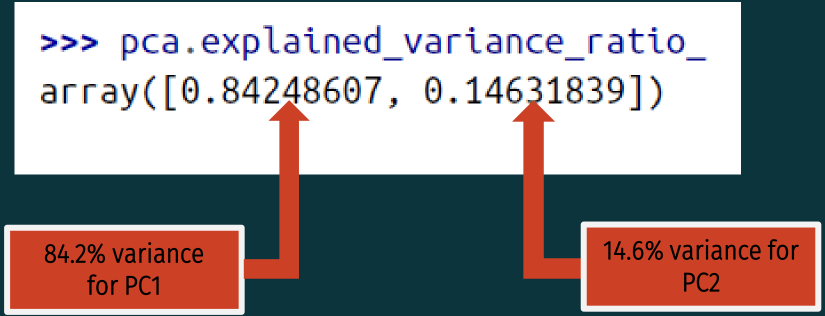


```
from sklearn.decomposition import PCA

pca = PCA(n_components = 2)
X2D = pca.fit_transform(X)
```

Explained Variance Ratio

- proportion of the dataset’s variance that lies along each principal component
- via the *explained_variance_ratio_* variable



```
>>> pca.explained_variance_ratio_
array([0.84248607, 0.14631839])
```

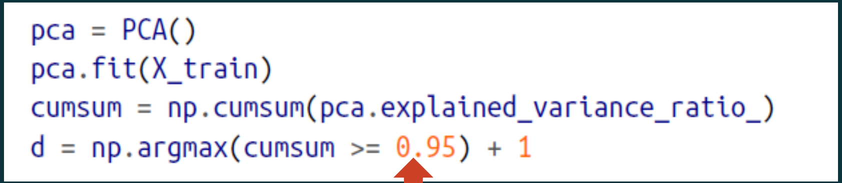
84.2% variance for PC1

14.6% variance for PC2

Source: Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow

CHOOSING THE RIGHT NUMBER OF DIMENSIONS

- choose the number of dimensions that add up to a sufficiently large portion of the variance (e.g., 95%)



```
pca = PCA()
pca.fit(X_train)
cumsum = np.cumsum(pca.explained_variance_ratio_)
d = np.argmax(cumsum >= 0.95) + 1
```

preserve 95% of variance

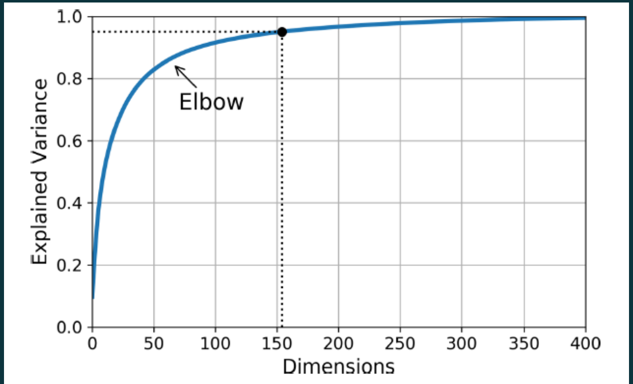
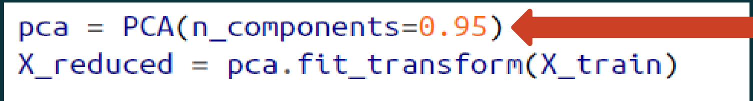


Figure: Explained variance as a function of the number of dimensions



```
pca = PCA(n_components=0.95)
X_reduced = pca.fit_transform(X_train)
```

Ratio of variance to preserve (lies between 0.0 and 1.0)

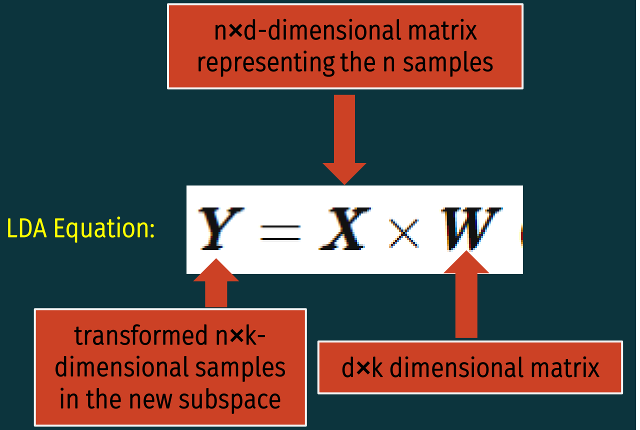
Source: Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow

PCA VERSUS LDA

- linear transformation techniques→ dimensionality reduction

Principal Component Analysis (PCA)

- ✓ Unsupervised
- ✓ Computes the principal components
- ✓ Maximize variance in dataset



LDA Equation:

$$Y = X \times W$$

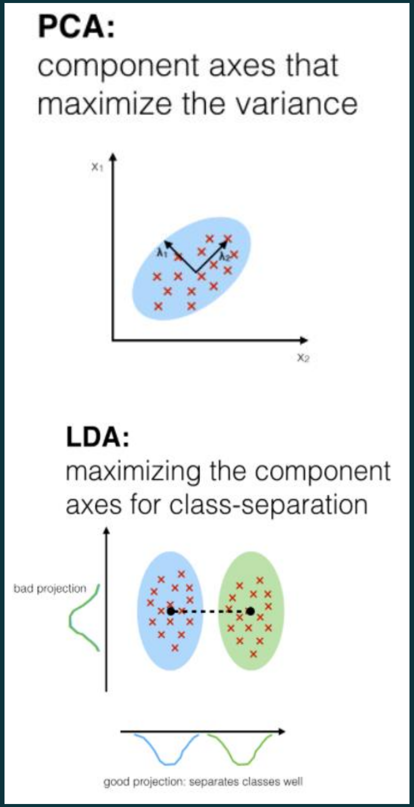
n×d-dimensional matrix representing the n samples

transformed n×k-dimensional samples in the new subspace

d×k dimensional matrix

Linear Discriminant Analysis (LDA)

- ✓ Supervised
- ✓ computes the directions→ linear discriminants
- ✓ project a n-dimensional dataset onto a smaller subspace k (where k≤n-1) while maintaining the class-discriminatory information
- ✓ Represent the axes that that maximize the separation between multiple classes



PCA:
component axes that maximize the variance

LDA:
maximizing the component axes for class-separation

Source: https://sebastianraschka.com/Articles/2014_python_lda.html

TRAINING AND EVALUATION

- Split the dataset into 2 or 3 parts: Train, Validation, Test
- Tuning of hyperparameters based on validation data→ information leaks into the model→ overfitting
- Performance on completely new data is important→ test data

Basic evaluation techniques:→

1. simple hold-out validation
2. Kfold validation
3. iterated K-fold validation with shuffling.

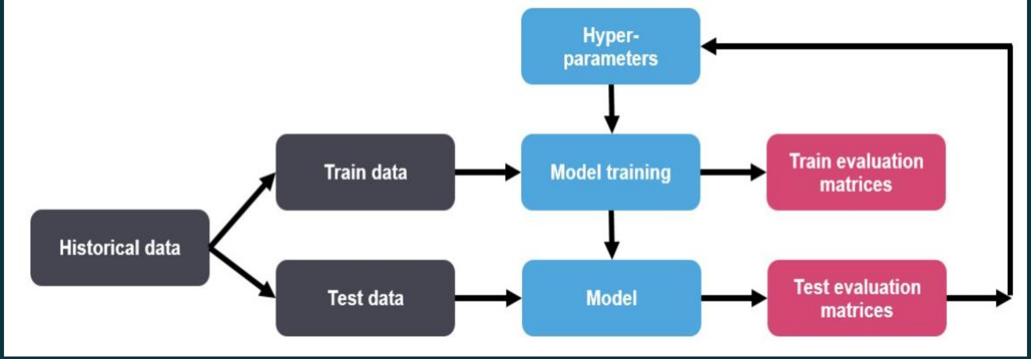
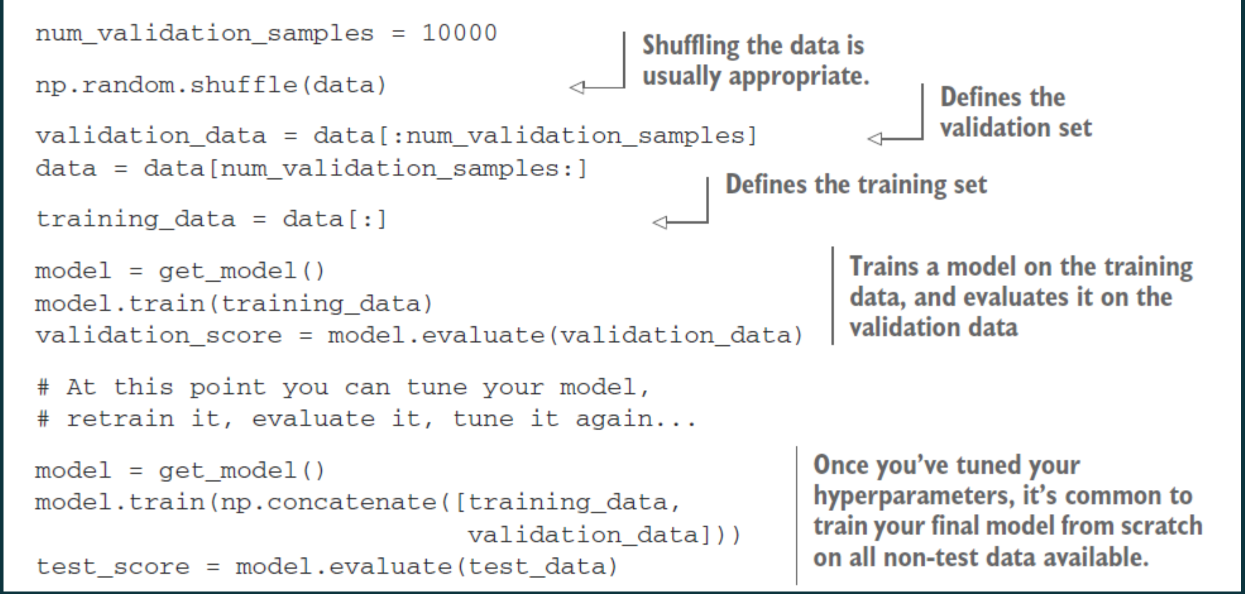


Image source: https://www.analyticsvidhya.com/

SIMPLE HOLD-OUT VALIDATION EXAMPLE



```
num_validation_samples = 10000

np.random.shuffle(data)

validation_data = data[:num_validation_samples]
data = data[num_validation_samples:]

training_data = data[:]

model = get_model()
model.train(training_data)
validation_score = model.evaluate(validation_data)

# At this point you can tune your model,
# retrain it, evaluate it, tune it again...

model = get_model()
model.train(np.concatenate([training_data,
                            validation_data]))
test_score = model.evaluate(test_data)
```

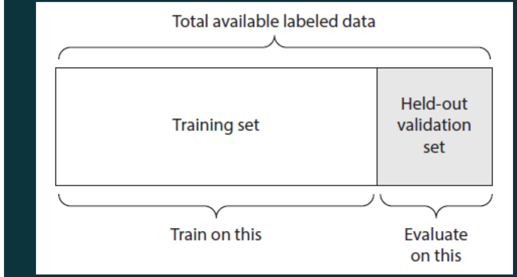
Shuffling the data is usually appropriate.

Defines the validation set

Defines the training set

Trains a model on the training data, and evaluates it on the validation data

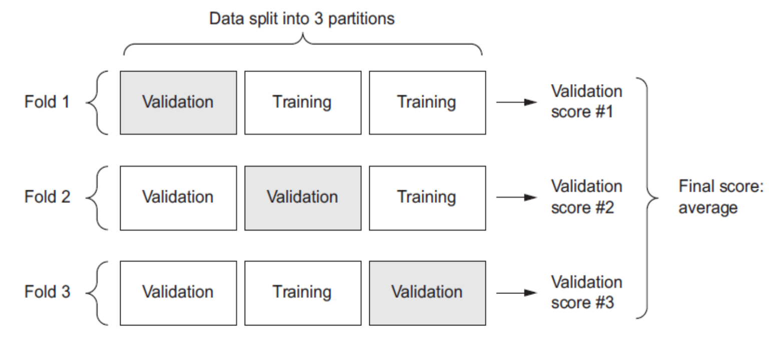
Once you've tuned your hyperparameters, it's common to train your final model from scratch on all non-test data available.

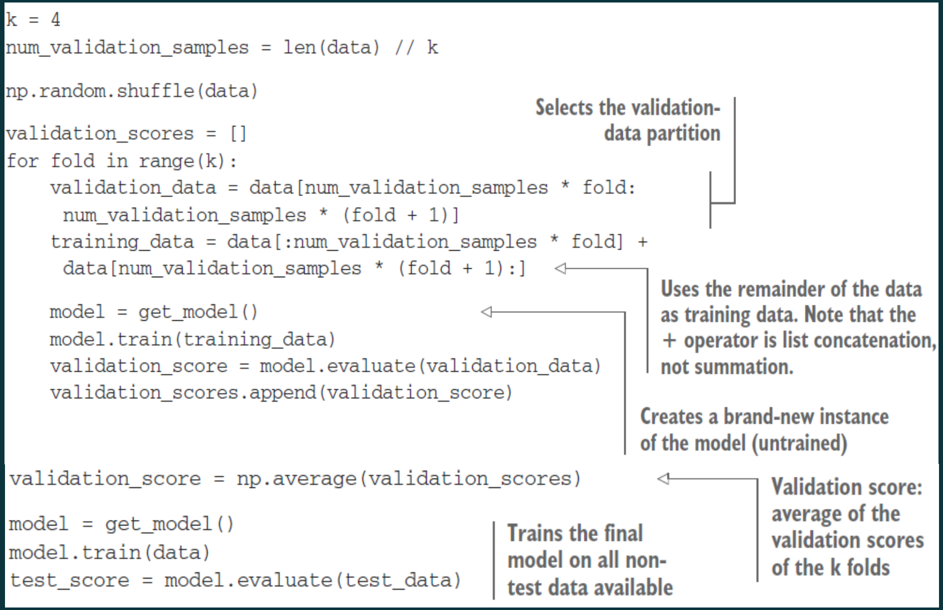


Source : Deep Learning with Python by François Chollet

K-FOLD CROSS VALIDATION EXAMPLE

- Split the dataset into K partitions of equal size
- For each partition i, train a model on the remaining K – 1 partitions, and evaluate it on partition i
- Final score is then the averages of the K scores obtained





```
k = 4
num_validation_samples = len(data) // k

np.random.shuffle(data)

validation_scores = []
for fold in range(k):
    validation_data = data[num_validation_samples * fold:
     num_validation_samples * (fold + 1)]
    training_data = data[:num_validation_samples * fold] +
     data[num_validation_samples * (fold + 1):]

    model = get_model()
    model.train(training_data)
    validation_score = model.evaluate(validation_data)
    validation_scores.append(validation_score)

validation_score = np.average(validation_scores)

model = get_model()
model.train(data)
test_score = model.evaluate(test_data)
```

Selects the validation-data partition

Uses the remainder of the data as training data. Note that the + operator is list concatenation, not summation.

Creates a brand-new instance of the model (untrained)

Validation score: average of the validation scores of the k folds

Trains the final model on all non-test data available

Source : Deep Learning with Python by François Chollet

ITERATED K-FOLD VALIDATION WITH SHUFFLING

- relatively little data available
- to evaluate your model as precisely as possible
- K-fold validation multiple times, shuffling the data every time before splitting it K ways
- final score is the average of the scores obtained at each run of K-fold validation
- High complexity→ P × K models (where P is the number of iterations)

Source : Deep Learning with Python by François Chollet

EVALUATION METRICS

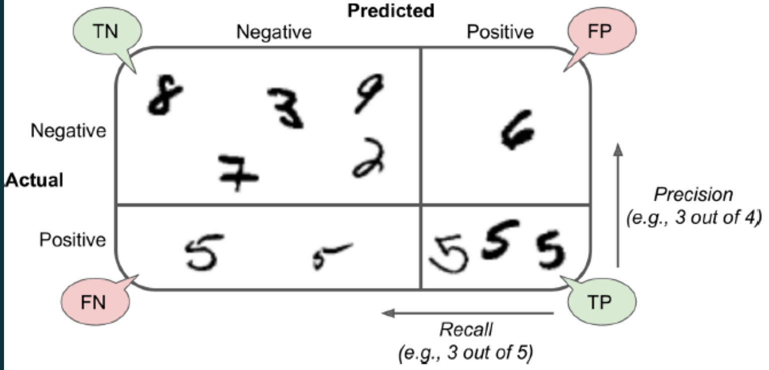
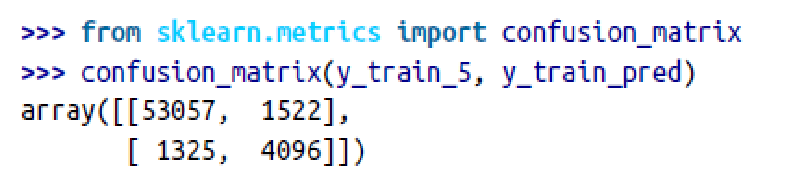


Figure: An illustrated confusion matrix for MNIST dataset shows examples of true negatives (top left), false positives (top right), false negatives (lower left), and true positives (lower right)



```
>>> from sklearn.metrics import confusion_matrix
>>> confusion_matrix(y_train_5, y_train_pred)
array([[53057,  1522],
       [ 1325,  4096]])
```

- True Positive (TP): Predicted True and True in reality
- True Negative (TN): Predicted False and False in reality
- False Positive (FP): Predicted True and False in reality, false alarm
- False Negative (FN): Predicted False and True in reality

Source: Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow

COMMON EVALUATION METRICS

Accuracy
ratio of number of correct predictions to the total number of input instances

$$\frac{\mathrm{TP}+\mathrm{TN}}{\mathrm{TP}+\mathrm{TN}+\mathrm{FP}+\mathrm{FN}}$$

Sensitivity or Recall or True Positive Rate
ratio of positive instances that are correctly considered as positive, with respect to all positive instances.

$$\mathrm{TPR}=\frac{\mathrm{TP}}{\mathrm{P}}=\frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}}$$

Specificity or True Negative Rate
ratio of negative instances that are correctly considered as negative, with respect to all negative instances

$$\mathrm{TNR}=\frac{\mathrm{TN}}{\mathrm{N}}=\frac{\mathrm{TN}}{\mathrm{TN}+\mathrm{FP}}$$

Positive Predictive Value or Precision
ratio of number of correct positive predictions to the total number of positive instances predicted by classifier

$$\mathrm{PPV}=\frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FP}}$$

F1 score
harmonic mean of precision and sensitivity

$$\mathrm{F}_1=2\times\frac{\mathrm{PPV}\times\mathrm{TPR}}{\mathrm{PPV}+\mathrm{TPR}}$$

False Positive Rate
ratio of negative instances that are mistakenly considered as positive, with respect to all negative instances

$$\mathrm{FPR}=\frac{\mathrm{FP}}{\mathrm{N}}=\frac{\mathrm{FP}}{\mathrm{FP}+\mathrm{TN}}$$

False Negative Rate
ratio of positive instances that are mistakenly considered as negative, with respect to all positive instances

$$\mathrm{FNR}=\frac{\mathrm{FN}}{\mathrm{P}}=\frac{\mathrm{FN}}{\mathrm{FN}+\mathrm{TP}}$$

Source: https://en.wikipedia.org/wiki/Evaluation_of_binary_classifiers

COMMON EVALUATION METRICS CONTD.

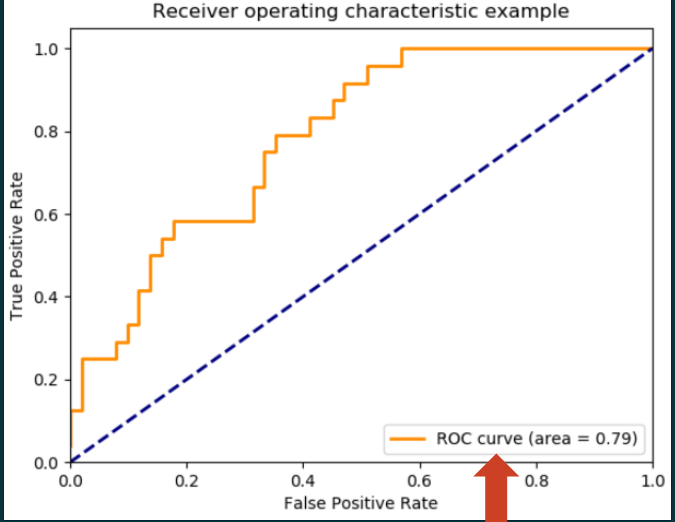
Receiver Operating Characteristic (ROC)
Graph between FPR and TPR

Mean Absolute Error (MAE)
average of the difference between the original values and the predicted values

$$MeanAbsoluteError = \frac{1}{N}\sum_{j=1}^{N}|y_j - \hat{y}_j|$$

Mean Squared Error (MSE)
average of the square of the difference between the original values and the predicted values

$$MeanSquaredError = \frac{1}{N}\sum_{j=1}^{N}(y_j - \hat{y}_j)^2$$



Area Under the Curve (AUC) has a range of [0,1]. Greater the value, better the model performance

Source: https://towardsdatascience.com/metrics-to-evaluate-your-machine-learning-algorithm-f10ba6e38234

PRECISION AND RECALL

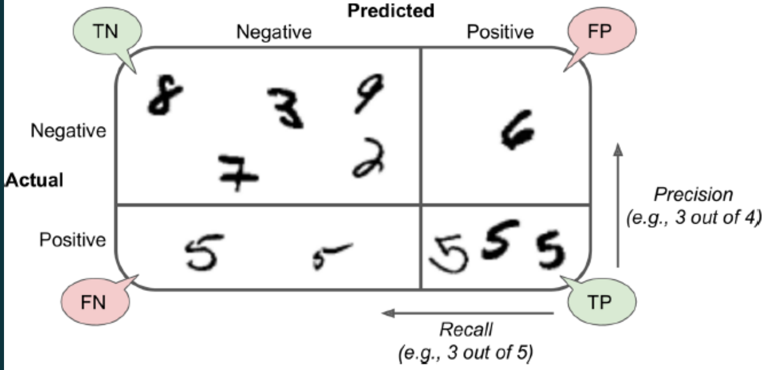
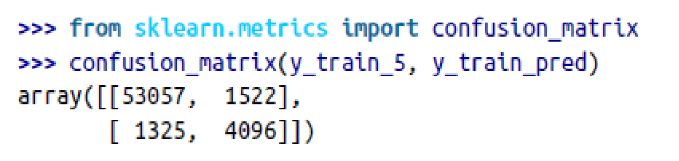


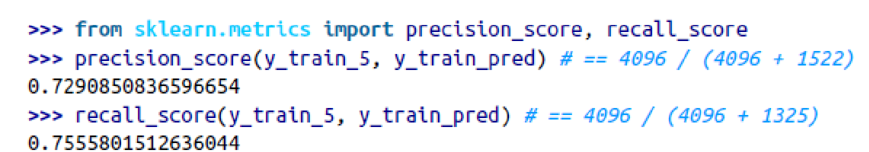
Figure: An illustrated confusion matrix for MNIST dataset shows examples of true negatives (top left), false positives (top right), false negatives (lower left), and true positives (lower right)

Display the Confusion Matrix



```
>>> from sklearn.metrics import confusion_matrix
>>> confusion_matrix(y_train_5, y_train_pred)
array([[53057,  1522],
       [ 1325,  4096]])
```

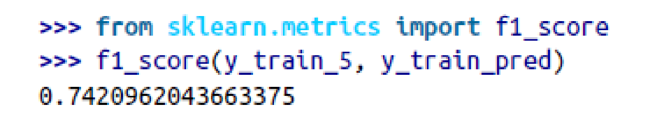
Calculating Precision & Recall



```
>>> from sklearn.metrics import precision_score, recall_score
>>> precision_score(y_train_5, y_train_pred) # == 4096 / (4096 + 1522)
0.7290850836596654
>>> recall_score(y_train_5, y_train_pred) # == 4096 / (4096 + 1325)
0.7555801512636044
```

F1 Score favors classifiers with similar precision and recall values

Calculating F1 Score



```
>>> from sklearn.metrics import f1_score
>>> f1_score(y_train_5, y_train_pred)
0.7420962043663375
```

Source: Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow

PRECISION/RECALL TRADE-OFF

- increasing precision reduces recall, and vice versa

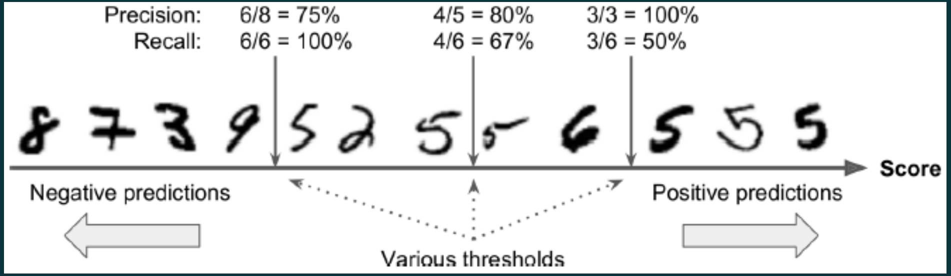


Figure: In this precision/recall trade-off, images are ranked by their classifier score, and those above the chosen decision threshold are considered positive; the higher the threshold, the lower the recall, but (in general) the higher the precision

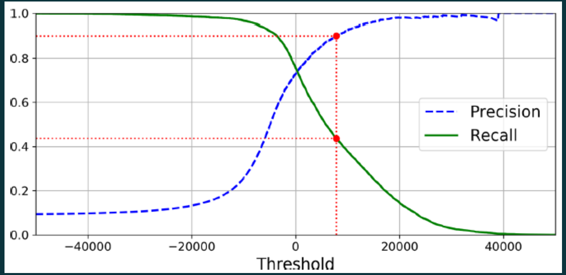


Figure: Precision and recall versus the decision threshold

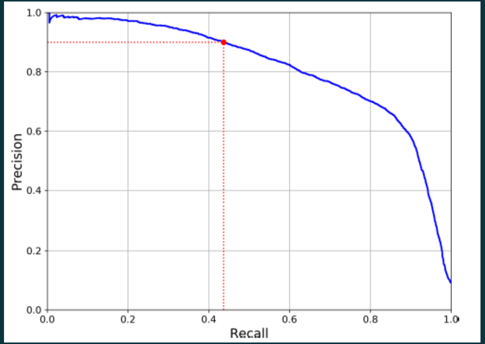


Figure: Precision versus recall

Source: Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow

HYPERPARAMETER OPTIMIZATION (HPO)

- According to Wikipedia "*hyperparameter optimization is choosing a set of optimal hyperparameters for a learning algorithm*"
- Hyperparameter→ parameter whose value is set before the learning process begins
- aim to find the right combination of their values to minimize loss or maximize accuracy

Examples:

- number of neurons learning rate or batch size for a neural network
- Penalty for Logistic Regression
- number of estimators in Random Forest

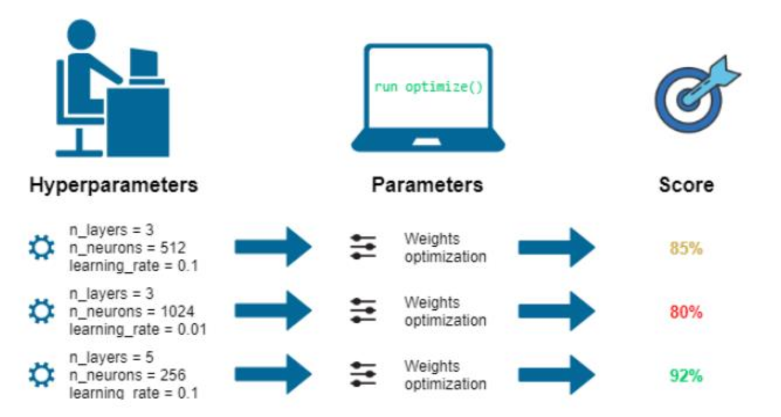
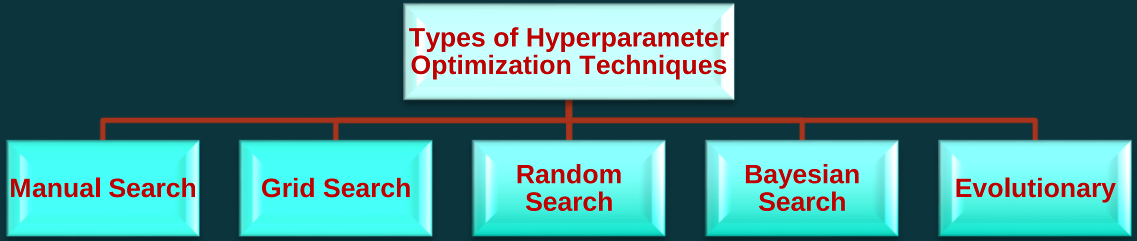


Image source: https://towardsdatascience.com/hyperparameters-optimization-526348bb8e2d

HYPERPARAMETER OPTIMIZATION (HPO)



Types of Hyperparameter Optimization Techniques

Manual Search

Grid Search

Random Search

Bayesian Search

Evolutionary

Manual Search→ choose hyperparameters based on our judgment, train, evaluate, repeat

Grid Search→ select a grid of hyperparameters and train/evaluate the AI with each possible combination, GridSearchCV class

Random Search→ select a grid of hyperparameters and train/evaluate the AI with random combinations, RandomizedSearchCV

Bayesian Search→

- automated HPO, for example, Hyperopt library, use probability to find the minimum of a function
- Final aim→ find input value for a function that gives the lowest possible output value (loss)

Evolutionary Search→ Genetic Algorithm, natural selection, automated HPO, e.g., TPOT

- Select population of N Machine Learning models with some predefined Hyperparameters→ calculate accuracy
- Keep the best performing models
- Repeat the process of selecting N ML models→ calculate accuracy
- Repeat for predefined generations→ until best models survive

HANDS-ON EXERCISES

THANKS!

Do you have any questions?