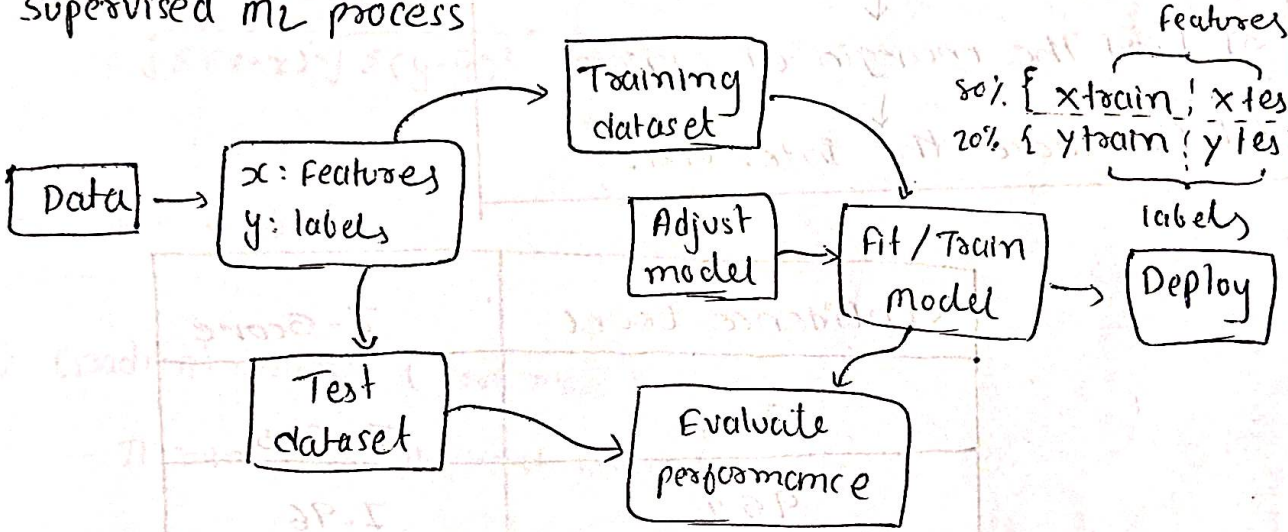
Machine learning

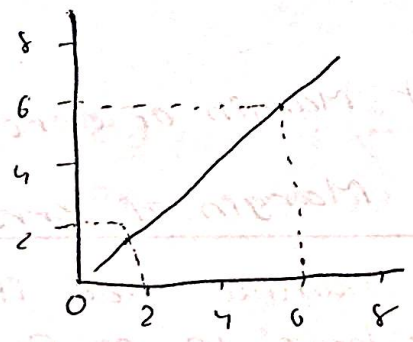
- Supervised
  - classification
  - regression
- Unsupervised → recognizing Pattern
- Reinforcement → Q-learning

1) Supervised learning

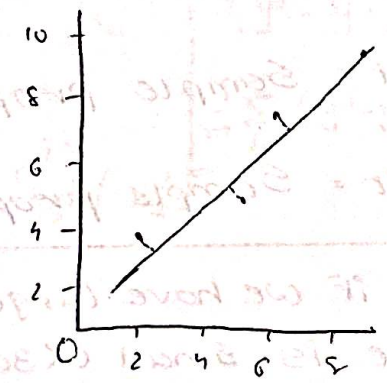
- Supervised ML process



i) linear regression



Here we can't predict ~~any~~ value of y for any x value.



line of best fit it is drawn such that it is at minimum distance from all the points

equation of line is:

→ $y = mx + c$

- m → slope
- c → y-intercept

⇒ $y = \beta_0 x_0 + \beta_1 x_1 + \dots \beta_n x_n = \sum_{i=1}^{n} \beta_i x$

There are 2 methods to calculate slope and intercept or we can say β values

1) ordinary least square (only for 1 feature 'x')

$$\hat{y} = b_1 x + b_0$$

$\Rightarrow b_1 = \rho_{xy} = \frac{\sigma_y}{\sigma_x}$ where $\rho_{x,y}$ = Pearson correlation coefficient

σ_x, σ_y = standard Deviation

$$b_1 = \frac{\sum (x-\bar{x})(y-\bar{y})}{\sqrt{\sum (x-\bar{x})^2 \sum (y-\bar{y})^2}} \cdot \frac{\sqrt{\frac{\sum (y-\bar{y})^2}{n}}}{\sqrt{\frac{\sum (x-\bar{x})^2}{n}}} = \boxed{\frac{\sum (x_i-\bar{x})(y_i-\bar{y})}{\sum (x_i-\bar{x})^2}}$$

$\Rightarrow b_0 = \bar{y} - b_1 \bar{x}$

2) Gradient descent (for more than 1 feature)

- It minimize the cost function.
- cost function is defined as

$$\Rightarrow J(\beta) = \frac{1}{2m} \sum_{i=1}^{m} (y^i - \hat{y}^i)^2$$

$$\Rightarrow \frac{\partial J}{\partial \beta_k}(\beta) = \frac{1}{m} \sum_{i=1}^{m} \left(y^i - \sum_{i=0}^{n} \beta_i x_i^j \right) (-x_k^j)$$

$\because y^i - \hat{y}^i \Rightarrow$ error in j row

$\because (y^i - \hat{y}^i)^2 \Rightarrow$ squared error

$\because \frac{1}{m} \sum_{i=1}^{m} (y^i - \hat{y}^i)^2 \Rightarrow$ avg. squared error of m rows

$$\Rightarrow \nabla_\beta J = \begin{bmatrix} \frac{\partial J}{\partial \beta_0} \\ \vdots \\ \frac{\partial J}{\partial \beta_n} \end{bmatrix}$$

gradient

Derivative of cost function with respect to each β

⇒ General code

1. import required libraries such as numpy, pandas & matplotlib
2. Read the dataset.
3. From sklearn.model_selection import train_test_split
4. X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=101)
5. From sklearn.linearmodel import LinearRegression
6. model = LinearRegression() # Here, we pass hyper parameter of model
7. model.fit(X_train, y_train)
8. model.predict(X_test)

⇒ Evaluating Regression

1. Mean absolute error (MAE)
2. mean squared error (MSE)
3. Root mean square error (RMSE)

1. MAE:

$$\frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

where y_i = actual value
$\hat{y}_i$ = predicted value

2. MSE

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

3. RMSE

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

8. test_prediction = model.predict(X_test)

9. from sklearn.metric import mean_absolute_error,
mean_squared_error

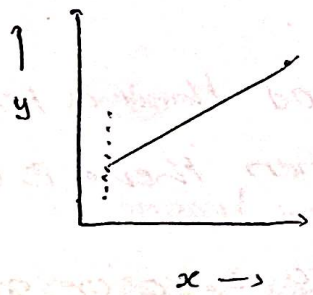
10. mean_absolute_error(y_test, test_prediction)

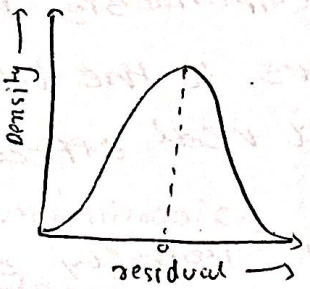
11. # RMSE
np.sqrt(mean_squared_error(y_test, test_predictions))

=> Evaluting Residuals

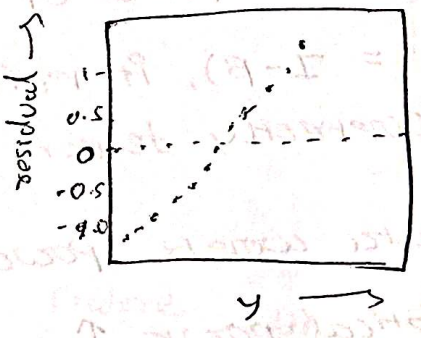
- $|y_i - \hat{y}_i|$ => Residuals

- plot the below 3 plots





normal dist.
(must be around '0' mean)



residual vs y plot
(point must be random)
- no curve
- no line
- no pattern

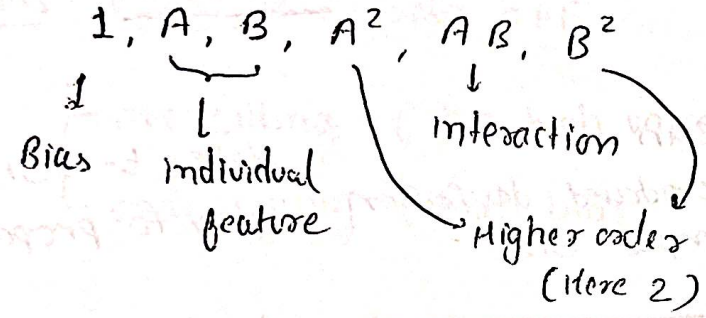
12. test_residuals = y_test - test_prediction
13. sns.scatterplot(x=y_test, y=test_residuals)

=> Model Deployement

14 Final_model = LinearRegression()
15 model.fit(X, y)
16 model.coef_ # return β value for all the features
17 from joblib import dump, load
18 dump(final_model, 'Final_sales_model.joblib') # saving
19 loaded_model = load('final_sales_model.joblib') # reload.

1.0) Polynomial Regression

- Sometimes A and B feature may not be that useful, but their interaction can be.
- So, we create new features from original feature using scikit learn PolynomialFeatures.
- Converting Two features A and B



⇒ Code:

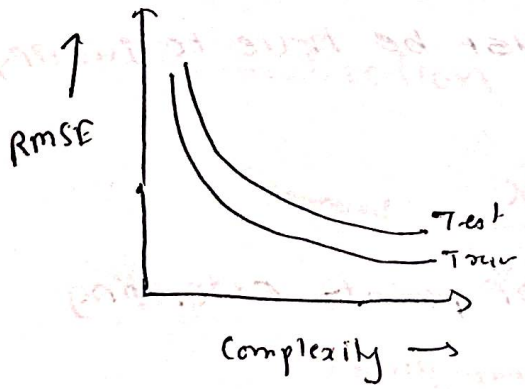
1. import important libraries
2. load / Read dataset
3. Identify features and label (X and y) — Feature, labels — [200, 3]
4. from sklearn.preprocessing import PolynomialFeatures
5. Polynomial_converter = polynomialFeatures (degree=2)
6. polynomical_converter.fit(X)
7. poly_feature = polynomical_converter.transform(X) [200, 9]
8. poly_feature = polynomical_converter.fit_transform(X)
9. from sklearn.model_selection import train_test_split
10. X_train, X_test, y_train, y_test = train_test_split (poly_feature, y, test_size=0.3, random_state=101)

```
11 ~~import~~ from sklearn.linear_model import LinearRegression
12 model = LinearRegression()
13 model.fit(X_train, y_train)
14 test_predict = model.predict(X_test)
15 From sklearn.metrics import mean_absolute_error,
                              mean_squared_error
16 MAE = mean_absolute_error(y_test, test_prediction)
```

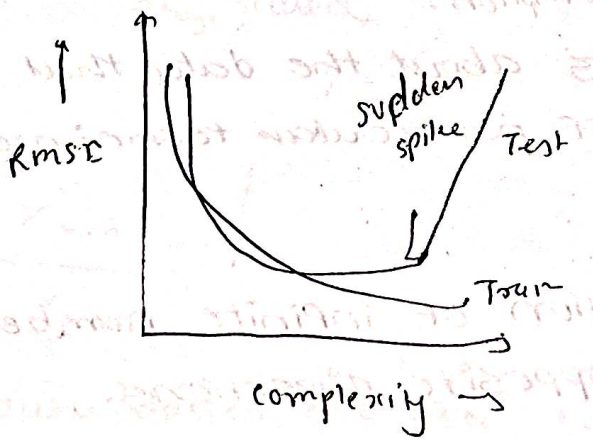
⇒ Bias - Variance Trade off

- overfitting (~~big~~ high variance)
- underfitting (high bias)

- Increasing the degree of polynomical increases the complexity which leads to overfitting problem.
- And lesser complexity than required creates problem of underfitting
- Hence, with very less or very high degree of polynomical RMSE increases



Ideal situation



overfitting at high complexity.

⇒ Regularization

1) Minimize model complexity
2) Penalize the loss function
3) Reduce model overfitting

- The 3 main types of regularization
  1) L1 Reg. (lasso regression)
  2) L2 Reg. (Ridge regression)
  3) Combine L1 and L2 (Elastic Net)

- Feature Scaling
  - If our data or features are on different scale, we use feature scaling
  - 2 main ways of scaling features:
    1. standardization (rescale data to have $\mu = 0$ & $\sigma = 1$) (mean, standard deviation)
    2. Normalization (rescale all data values bet^n 0-1)

1. Standardization:

$$X_{changed} = \frac{X - \mu}{\sigma}$$

X - original
$X_{changed}$ scaled

2. Normalization

$$X_{changed} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

- Fit & transform data classes are used for scaling
  - Fit: simply calculates the necessary statistics
  - transform: actually scales data and return new scaled data

- Feature scaling process

1. Perform train test split
2. Fit to training feature data
3. Transform training feature data
4. Transform test feature data.

=> code

1. From sklearn.preprocessing import standard Scaler
2. Scaler = standard Scaler()
3. Scaler.fit(X_train)
4. train_scale = scaler.transform(X_train)
5. test_scale = scaler.transform)(X_test)

1) L2 Regularization or Ridge Regression

- general formula for regression line

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2 + \dots + \hat{\beta}_p x_p$$

- These β coefficient were solved by minimizing RSS (Residual sum of squares)

$$\Rightarrow RSS = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

$$\Rightarrow = \sum_{i=1}^{n} (y_i - \hat{\beta}_0 x_{i1} + \hat{\beta}_1 x_{i2} \dots \hat{\beta}_p x_{ip})^2$$

$$= \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij}\right)^2 + \boxed{\lambda \sum_{j=1}^{p} \beta_j^2} \rightarrow \text{Penalty term}$$

- Ridge regression minimize RSS + penalty term

=> code:

1. From sklearn import Ridge
2. ridge_model = Ridge (alpha = 10) → Hyperparameter
3. ridge_model.fit (x_train, y_train)
4. test_predict = ridge_model.predict (x_test)
5. Calculate error MAE & RMSE

Ridge regression with cross validation

6. From sklearn import RidgeCV
7. ridge_cv_model = RidgeCV (alphas = [0.1, 1.0, 10.0], scoring = 'error string')
8. ridge_cv_model.fit (x_train, y_train)
9. ridge_cv_model.alpha_ # gives best performing alpha value
10. test_prediction = ridge_cv_model.predict (x_test)
11. MAE = mean_absolute_error (ytest, test_predictions)

2) L1 Regularization or LASSO regression

- It adds penalty equal to the absolute value of the magnitude of coefficient
- It can yield sparse model where some coefficient can be zero
- Error = $\sum_{i=1}^{n} (y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij})^2 + \lambda \sum_{j=1}^{p} \beta_j$ → Penalty term
- LASSO - least absolute shrinkage and selection operator

=> code

1. from sklearn.linear_model ~~from~~ import LassoCV
2. lasso_cv_model = LassoCV (eps = 0.001, n_alphas = 100, cv=5)
3. lasso_cv_model.fit (x_train, y_train)
4. lasso_cv_model.alpha_
5. test_prediction = lasso_cv_model.~~y_test~~ predict (X_test)
6. calculate the error

=> Elastic net

- $$\underset{\beta}{\text{minimize}} \left\{ \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 \right\} \text{ subject to } \sum_{j=1}^{p} |\beta_j| \leq s$$

and

$$\underset{\beta}{\text{minimize}} \left\{ \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 \right\} \text{ subject to } \sum_{j=1}^{p} \beta_j^2 \leq s$$

- For 2 features

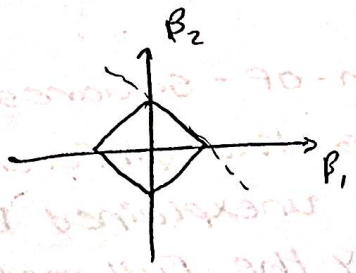
$$\hat{y} = \beta_1 x_1 + \beta_2 x_2$$

Lasso regression

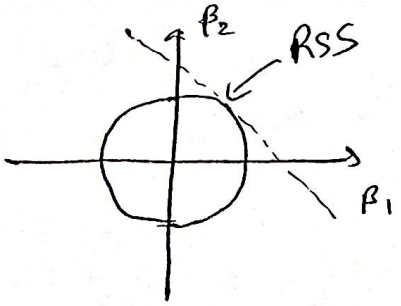
$$|\beta_1| + |\beta_2| \leq s$$

Ridge regression

$$\beta_1^2 + \beta_2^2 \leq s$$



lasso



- Error $= \sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2 + \lambda_1 \sum_{j=1}^{p}\beta_j^2 + \lambda_2 \sum_{j=1}^{p}|\beta_j|$

$$= \frac{\sum_{i=1}^{n}(y_i - x_i^T\hat{\beta})^2}{2n} + \lambda\left(\frac{1-\alpha}{2}\sum_{j=1}^{m}\hat{\beta}_j^2 + \alpha\sum_{j=1}^{m}|\hat{\beta}_j|\right)$$

⇒ code

1. From sklearn.linear_model import ElasticNetCV
2. Elastic_model = ElasticNetCV(l1_ratio = [.1, .5, .7, .9, .95], eps = 0.001, n_alphas = 100, max_iter = 1000000)
3. Elastic_model.fit(x_train, y_train)
4. Elastic_model.l1_ratio_ # gives the best l1_ratio
5. test_prediction = elastic_model.predict(x_test)
6. Calculate the error

* Feature Engineering

- Extracting features from raw data via data mining techniques.
- 3 approach to perform feature Engineering
  1. Extracting info.
  2. Combining info.
  3. Transforming ~~dtg~~ info.
     - [Integer Encoding (convert categories into 1,2,3.. N)
     - One-hot Encoding (convert each categories into 0 & 1]

* Data cleaning

1) Dealing with outliers:
   - outlier consideration depends on
     1. Range and limit
     2. Percentage of Data
   - Highly dependent on Domain
   - Outlier with same methodology
     - Inter Quartile Range
     - Standard Deviation
     - Visualized
   - $IQR = Q_3 - Q_1$

     lower-limit = $Q_1 - 1.5 * IQR$

     Upper-limit = $Q_3 + 1.5 * IQR$

code:

```
q75, q25 = np.percentile(df, [75, 25]) # return array of Q3 & Q1
IQR = q75 - q25
```

2) Missing value imputation:

- Missing values can be NAN, NULL, 0 etc

- code:

1. import libraries
2. import Dataset as df
3. df = ds.drop("PID", axis=1) # Drop unique column
4. 100* df.isnull().sum()/len(df) # percentage of null values
5. def percent_missing(df):
   percent_nan = 100*df.isnull().sum()/len(df)
   percent_nan = percent_nan[percent_nan>0].sort_values()
   return percent_nan
   Percent_nan =
6. percent_missing(df) # gives the % of null values in all column.
7. sns.barplot(x = percent_nan.index, y = percent_nan)
   plt.xticks(rotation = 90) # optional to visualize the null values

8. If the percentage of null values is less than 1%. then, we will drop or bill few rows missing the data feature

9. percent_nan[percent_nan < 1] # gives all the feature whose % is < 1
10. df[ds['Electrical'].isnull()] # gives the rows which has Electrical as null value

11. Now, If the feature column has 'NaN' or '0' for no data registered, we can fill na with respective values else, we have to delete the row

12. df = df.dropna(axis=0, subset=['Electrical', 'GarageCars'])

13. # NUMERIC COLUMNS --> Fill na 0

```
num_cols = ['col_name1', 'col_name2']
df[num_cols] = df[num_cols].fillna(0)
```

STRING COLUMNS --> Fill na 'none'

```
str_cols = ['col_name1', 'col_name2']
df[str_cols] = df[str_cols].fillna('None')
```

14. Now, If the percentage of null values is greater than 1% then, we will drop or fill columns. If percentage is greater than 50, we will drop all those columns

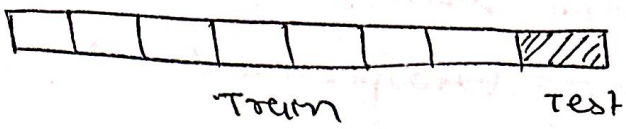
15. `df = df.drop(['poolQC', 'MiscFeature', 'Alley'])`

16. If column has no data and is filled with NA or 0, I will fill na with 'NA' or '0' for respective columns. And for other numeric column, we will fill na with some statistical computation.

17. `df['GarageYrBlt'] = df['GarageYrBlt'].fillna(0)`

18. `df.groupby('Neighborhood')['LotFrontage'].transform(lambda value : value.fillna(value.mean()))`

* Cross validation



Error general Train / Test split

=> Train / Test split procedure

- clean and adjust data for x and y
- split data in Train / Test split for both x and y
- Fit / Train scaler on Training x Data
- Scale x Test Data
- Create model
- Fit / Train model on x Train
- Evalute model on x Test data

=> code

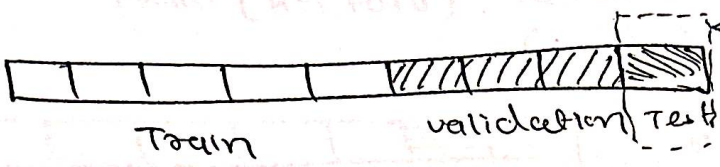
1. X = df.drop ['sales']
2. y = df ['sales']
3. X_train, X_test, y_train, y_test = train_test_split (X, y, test_size = 0.3, random_state
4. From sklearn.preprocessing import StandardScaler
5. Scaler = StandardScaler()
6. Scaler.fit (X_train)
7. X_train = scaler.transform (X_train)
8. X_test = scaler.transform (X_test)
9. From sklearn.linear_model import Ridge
10. model = Ridge (alpha =)
11. model.fit (X_train, y_train)

12. y_pred = model.predict(X_test)

13. from sklearn.metric import mean_squared_error

14. MSE = mean_squared_error(y_test, y_pred)

=> Train, validation & Test split



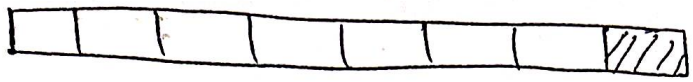
- we will do this by performing train_test_split() twice

  1. once split obb larger training set.
  2. remaining data into a validation & test set

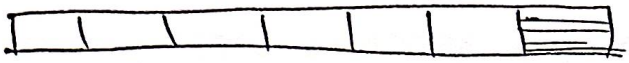
- code: # 70 : 15 : 15 (similar to general split)
  T V T

```
• X_train, X_other, y_train, y_other = train_test_split(X, y
                                        test_size=0.3,
                                        random_state=101)

• X_eval, X_test, y_eval, y_test = train_test_split(
                                    X_other, y_other,
                                    test_size=0.5,
                                    random_state=101)

  .
  .
  .
• X_eval = scaler.transform(X_eval)
  .
  .
  .
```

=> K-Fold cross validation

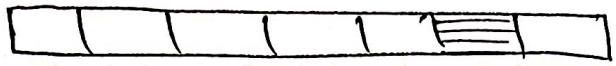


Test



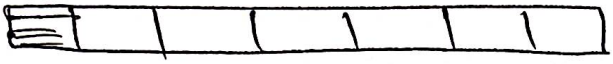
error 1

Train (K-1 Fold) validate (1 Fold)



error 2

& upto all possible folds.



error 5

---

avg error

= codes

~~1. df = df['sales'],~~

1. X = df.drop ['sales', axis = 1]

2. y = df ['sales']

3. x_train, x_test, y_train, y_test = train_test_split(X,
test_size = 0.3,
random_state = 101)

4. From sklearn.preprocessing import StandardScaler

5. scaler = StandardScaler()

6. scaler.fit (x_train)

7. X_train = scaler.transform (x_train)

8. X_test = scaler.transform (y_test) 9. model = Ridge (alpha=1,

10. From sklearn.model_selection import cross_val_score

11. scores = cross_val_score (model, x_train, y_train,
scoring = 'neg_mean_squared_error',
cv = 5)

```
12. abs (scores.mean()) # report avg error
13. model.fit (X_train, y_train)
14. y_final_pred = model.predict (X_test)
15. mean_squared_error (y_test, y_final_pred)
```

- We can also use cross_validate() function similarly as cross_val_score(), it just gives more information like fit_time and score_time.

* Grid search

- A grid search is a way of training and validating a model on every possible combination of multiple hyperparameter options.

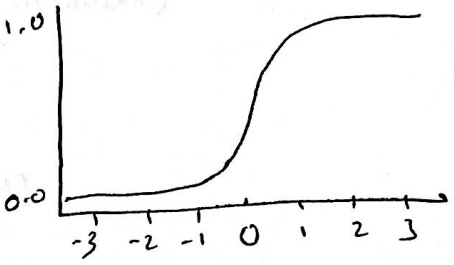
=> code

```
1. create X and y
2. Train and Test split
3. Scale data
4. from sklearn.linear_model import ElasticNet
5. base_elastic_model = ElasticNet()
6. param_grid = {'alpha': [0.1, 1, 5, 10, 50, 100],
                 'l1_ratio': [.1, .5, .7, .95, .99, 1]}
7. from sklearn.model_selection import GridSearchCV
8. grid_model = GridSearchCV (estimator = base_elastic_mod
                              param_grid = param_grid,
                              scoring = 'neg_mean_squared_er
                              cv=5, verbose=2)
9. grid_model.fit (X_train, y_train)
```

2) Logistic Regression

- It is a classification algorithm. It allows us to predict a categorical label based on historical feature data.
- The categorical column (label) may have two or more discrete class labels.
- logistic function or sigmoid function

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



→ $\hat{y} = \frac{1}{1+e^{-\sum_{i=1}^{n}\beta_i x_i}}$ $\left[\because \hat{y}(x) = \beta_1 x_1 + \beta_2 x_2 + \dots \beta_n x_n \right.$

$\left. \sigma(x) = \frac{1}{1+e^{-x}} \therefore \text{putting } x = \hat{y}(x) \right]$

$$\Rightarrow \hat{y} + \hat{y}e^{-\sum_{i=1}^{n}\beta_i x_i} = 1$$

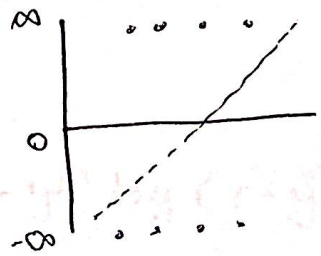
$$\Rightarrow \hat{y}e^{-\sum_{i=1}^{n}\beta_i x_i} = 1 - \hat{y}$$

$$\Rightarrow e^{-\sum_{i=1}^{n}\beta_i x_i} = \frac{1-\hat{y}}{\hat{y}}$$

$$\Rightarrow e^{\sum_{i=1}^{n}\beta_i x_i} = \frac{\hat{y}}{1-\hat{y}} \rightarrow \text{odds of log}$$

[odds is ratio of probability of event occuring to probability of event not occuring]

$$\Rightarrow \ln\left(\frac{\hat{y}}{1-\hat{y}}\right) = \sum_{i=0}^{n}\beta_i x_i$$



⇒ 1st step for maximum likelihood is to go from log odds back to probability

$$\Rightarrow \ln\left(\frac{p}{1-p}\right) = \ln(\text{odds})$$

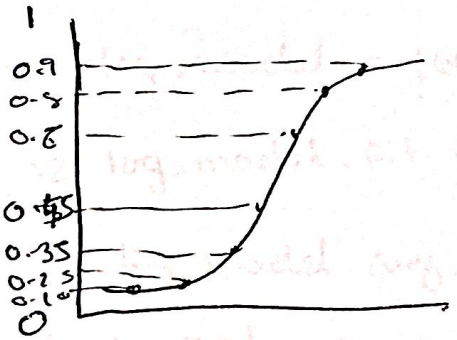
$$\Rightarrow \frac{p}{1-p} = e^{\ln(\text{odds})}$$

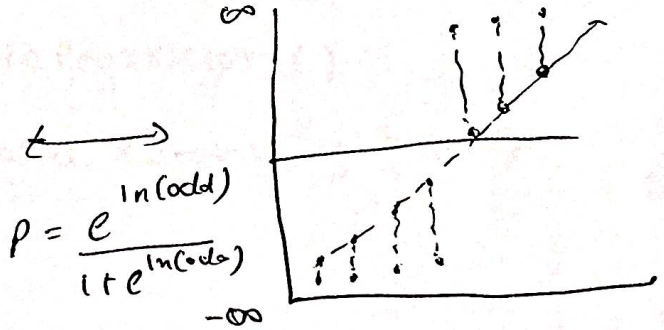
$$\Rightarrow p = e^{\ln(\text{odds})} - p e^{\ln(\text{odds})}$$

$$\Rightarrow p + p e^{\ln(\text{odds})} = e^{\ln(\text{odds})}$$

$$\Rightarrow p = \frac{e^{\ln(\text{odds})}}{1 + e^{\ln(\text{odds})}}$$

⇒ We, are trying to maximize the likelihood





so likelihood = $0.9 \times 0.8 \times 0.6 \times (1-0.45) \times (1-0.35) \times (1-0.25) \times (1-0.10)$

maximize this by try different line of best fit.

⇒ In terms of cost function, we seek to minimize the following (log loss):

$$J(x) = -\frac{1}{m}\sum_{i=1}^{m} y^i \log(\hat{y}^i) + (1-y^i)\log(1-\hat{y}^i)$$

⇒ code

1. import important libraries
2. X = df.drop ('test_result', axis=1)
3. y = df['test_result']
4. From sklearn.model_selection import train_test_split
5. From sklearn.preprocessing import StandardScaler
6. X_train, X_test, y_train, y_test = train_test_split (X, y, test_size = 0.1, random_state = 101)
7. Scaler = StandardScaler ()
8. scaled_x_train = scaler.fit_transform (x_train)
9. scaled_x_test = scaler.transform (x_test)
10. From sklearn.linear_model import logisticRegression ()
11. log_model = logisticRegression ()
12. log_model.fit (scaled_x_train, y_train)
13. log.model.coef_
14. y_pred = log_model.predict (scaled_x_test)

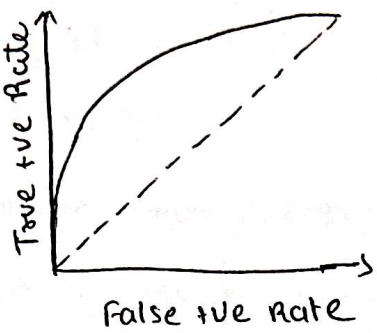
⇒ clasification evalution metrics

- Confusion matrix

Predicted \ Actual	Infected	Healthy
Infected	True +ve	False +ve
Healthy	False -ve	True -ve

- Accuracy = $\frac{(TP+TN)}{Total}$
- Recall = (TP) / Total actual +ve
- Precision = (TP) / Total predicted +ve
- F1 score = $\frac{2 \times Precision \times Recall}{Precision + Recall}$

- ROC curve : Receiver Operator characteristic curve



- we gladly accept more false +ve to reduce false negatives

=> code :

1. from sklearn.metrics import accuracy_score, confusion_matrix, classification_report.
2. y_pred = log_model.predict(scaled_x_test)
3. accuracy_score(y_test, y_pred)
4. confusion_matrix(y_test, y_pred)
5. from sklearn.metrics import plot_confusion_matrix
6. plot_confusion_matrix(log_model, scaled_x_test, y_test)
7. ~~class~~ print(classification_report(y_test, y_pred)) # gives precision, recall, f1-score and support.
8. precision_score() # Similarly as confusion matrix
   recall_score() # " " "
9. from sklearn.metrics import plot_precision_recall_curve, plot_roc_curve
10. plot_roc_curve(log_model, scaled_x_test, y_test)
11. plot_precision_recall_curve(log_model, scaled_x_test, y_test)
12. log_model.predict_proba(scaled_x_test) # get probabilities % belong class

* Multiclass logistic Regression

1. import libraries
2. create x and y
3. Create train / test split
4. Fit transform train } scale data
5. transform test }
6. From sklearn.linear_model import LogisticRegression
7. from sklearn.model_selection import GridSearchCV
8. log_model = LogisticRegression (solver='saga', multi_class='ovr', max_iter=5000)
9. penalty = ['l1', 'l2', 'elasticnet']
   l1_ratio = np.linespace(0, 1, 20)
   C = np.logspace(0, 10, 20)
   param_grid = {'penalty': penalty, 'l1_ratio': l1_ratio, 'C': C}
10. grid_model = GridsearchCV(log_model, param_grid = param_g
11. grid_model.fit(scaled_x_train, y_train)
12. Evaluate using matrics.

3) KNN

- It assign a label to new data based on the distance bet^n the old data and new data
- K is the no. of distance point, we considere to label new data point
- The label which has major no. of shortest distance the new point gets assigned to that label.
- Eg for K = 5, 3 distance are shortest to class A data point
  2 distance are shortest to class B data point

  then the new data point belongs to Class A.
- We want K value that minimizes error

  error = 1 - Accuracy
- KNN considerations for distance metric
  - minkowski
  - Euclidean
  - manhattan
  - chebyshev
- Things to keep in mind for KNN algorithm.
  - choosing optimal K value.
  - Scaling features.

=> code

1. import important libraries
2. import dataset
3. creat X and y
4. Creat train_test_split
5. scale data
6. From sklearn.neighbors import KNeighborsClassifier
7. knn_model = KNeighborsClassifier()
8. knn_model.fit (scaled_x_train, y_train)
9. y_pred = knn_model.predict (scaled x_test)
10. Evalute errors # confusion metrics and classification report.

=> Pipeline

- Pipeline object can set up a sequence of repeated operations such as a scaler and a model.

=> code

1. Scaler = StandardScaler()
2. knn = KNeighborsClassifier()
3. knn.get_params().keys() # strange code for parameters
4. operations = [('scaler', scaler), ('knn', knn)]
5. From sklearn.pipeline import Pipeline
6. Pipe = Pipeline (operations)
7. From sklearn.model_selection import GridSearchCV
8. k_values = list (range (1,20))
9. param_grid = ~~{'n_neighbors': k_values}~~
   {'knn__n_neighbors': k_values}

4) Support Vector Machines

Maximum margin classifier

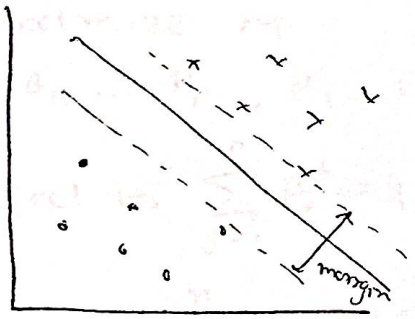
↓

support vector classifier (SVC) (add bias to allow some miss-classificati

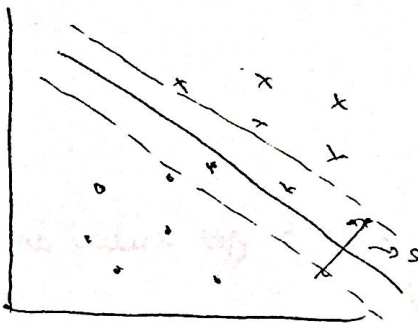
↓

support vector machines (SVM)

- For N dimensioned space, a hyperplane is N-1 dimension
  - 1-D Hyperplan is a single point
  - 2-D " " " line
  - 3-D " " " plane

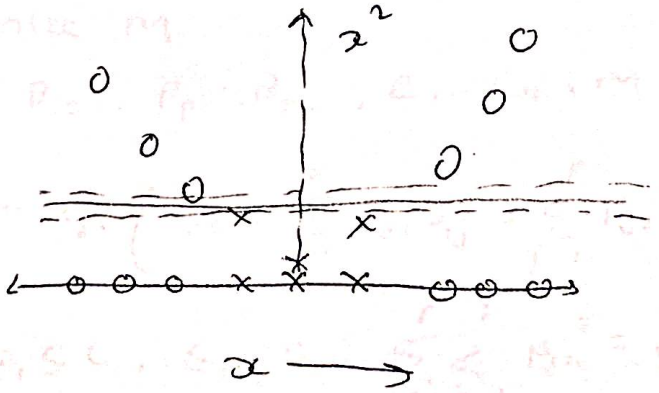


maximum margin



SVC → soft margin that allow some miss-classificat

- SVMs use kernel to project the data into higher Dimensi in order to use hyper plane.



=> Mathematics of ~~Hessel trick~~

- $x_1 = \begin{pmatrix} x_{11} \\ x_{12} \\ \vdots \\ x_{1p} \end{pmatrix}, \ldots, x_n = \begin{pmatrix} x_{n1} \\ x_{n2} \\ \vdots \\ x_{np} \end{pmatrix}$

- maximize M

  $\beta_0, \beta_1, \ldots, \beta_p, M$

  subject to $\sum_{j=1}^{p} \beta_j^2 = 1$

- $y_i(\beta_0 + \beta_1 x_{i1} + \ldots + \beta_p x_{ip}) > M \quad \forall i = 1 \ldots n$

=> Support vector classifier

- maximize M

  $\beta_0, \beta_1, \ldots \beta_p, \epsilon_1 \ldots \epsilon_n, M$

  subject to $\sum_{j=1}^{p} \beta_j^2 = 1$

- $\epsilon_i \geq 0, \sum_{i=1}^{n} \epsilon_i \leq C$ → Higher the value of C less bias (MMC)
  → lower (SVC)

- $y_i(\beta_0 + \beta_1 x_{i1} + \ldots \beta_p x_{ip}) \geq M(1 - \epsilon_i)$

=> support vector machines

$x_1, x_1^2, x_2, x_2^2, \ldots, x_p, x_p^2$

maximize M

$\beta_0, \beta_{11}, \beta_{12} \ldots \beta_{p1}, \beta_{p2}, \epsilon_1 \ldots \epsilon_n, M$

subject to $y_i\left(\beta_0 + \sum_{j=1}^{p} \beta_{j1} \beta_{ij} + \sum_{j=1}^{p} \beta_{j2} x_{ij}^2\right) \geq M(1 - \epsilon_i)$

$\sum_{i=1}^{n} \epsilon_i \leq C, \quad \epsilon_i \geq 0, \quad \sum_{j=1}^{p} \sum_{k=1}^{2} \beta_{jk}^2 = 1$

=> To deal with very large feature space we use kernel trick, which uses the inner product of vectors (dot product)

- Dot product

$$\langle a,b \rangle = \sum_{i=1}^{r} a_i b_i$$

$$a.b = a_1 b_1 + a_2 b_2$$

$$a.b = |a||b|\cos\theta$$

- SVC rewritten as

$$f(x) = \beta_0 + \sum_{i=1}^{n} \alpha_i \langle x, x_i \rangle$$

↓

$$f(x) = \beta_0 + \sum_{i \in S} \alpha_i \langle x, x_i \rangle$$

- Kernel function

$$K(x_i, x_{i'}) = \sum_{j=1}^{p} x_{ij}\, x_{i'j}$$

$$\langle a,b \rangle = \sum_{i=1}^{r} a_i b_i$$

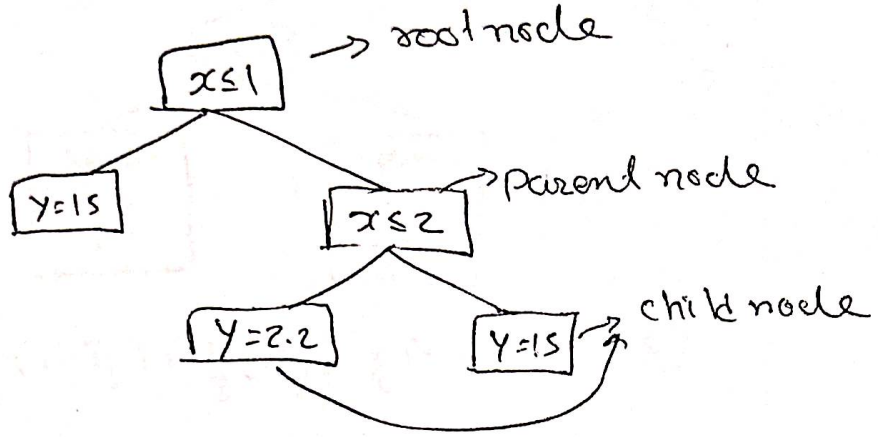
$$f(x) = \beta_0 + \sum_{i \in S} \alpha_i \langle x, x_i \rangle$$

=> code:

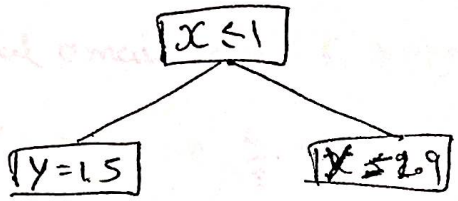
1. From sklearn.svm import SVC
2. create X and y
3. model = SVC (kernel = 'linear', C=1000) # C is inversly proportional to the error.
   model.fit (X, y)
4. From svm_margin_plot import plot_svm_boundary
5. plot_svm_boundary (model, X, y)

   # A low C make decision surface smooth, i.e makes soft margin. whereas high C value makes maximum margin. low C value is better.

Decision Tree



⇓ pruning (to short the length of Tree to avoid overfitting)

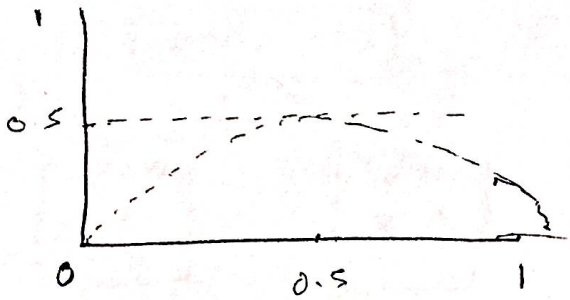


⇒ Gini impurity

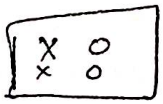
- It is measurement of how 'pure' the information in a dataset is.

- $G(Q) = \sum_{c \in C} P_c(1 - P_c)$

  (P_c: probability of class c; Q: dataset; C: classes)



- Eg



P class A

$\frac{2}{4}(1 - \frac{2}{4}) = 0.25$

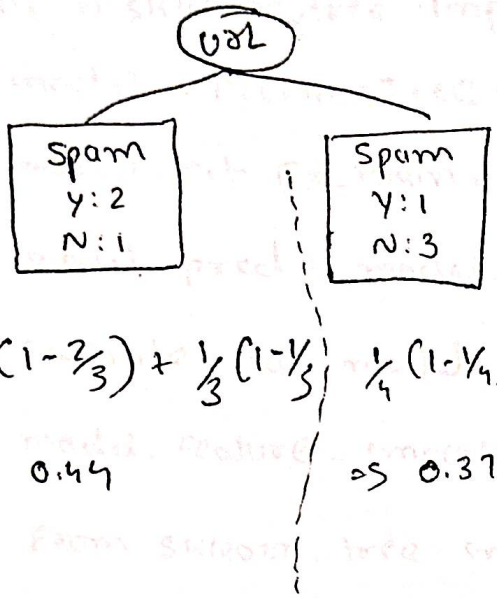
P class B

$\frac{2}{4}(1 - \frac{2}{4}) = 0.25$

∴ gini impurity = 0.25 + 0.25 = 0.50

- We went to minize the gini impurity at leaf nodes.

⇒ Gini impurity for binary categorical feature



$\frac{2}{3}(1-\frac{2}{3}) + \frac{1}{3}(1-\frac{1}{3})$ $\quad$ $\frac{1}{4}(1-\frac{1}{4}) + \frac{3}{4}(1-\frac{3}{4})$

⇒ 0.44 $\quad$ ⇒ 0.375

- Total emails = 7 (3 left Emails & 4 Right Emails)

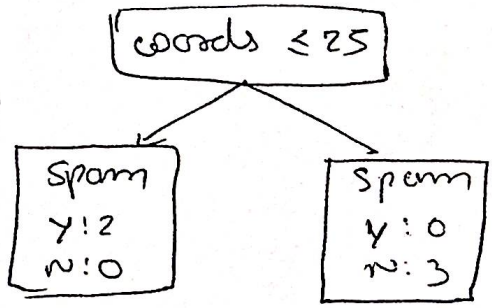
$$\frac{3}{7} \times 0.44 + \frac{4}{7} \times 0.375$$

⇒ 0.375

⇒ Gini impurity for continous feature

x-coords	y-spam	
10	yes	Gini = 0.3
15		
20	yes	Gini = 0
25		
30	NO	Gini = 0.26
35		
40	NO	Gini = 0.4
45		
50	NO	

⇒



- Choose the best split value such that we get lowest gini.

Random Forest (RF can not be overfitted)

- Random forest is a decision tree made on different subset of features.
- Unique Random forest hyperparameters:
  1. no. of Estimators - How many DT to use in forest
  2. no. of Features - How many Feature to be include in each
  3. Bootstrap samples - Allow bootstrap sampling of each training subset
  4. Out of Bag Error - calculate OOB error during training.

1. no. of Estimators: 64 - 128 DT are good, we can use as may DT as we want. There is no overfitting. Trees just gets duplicated. Start with 'two'.

2. no of feature: According to original publication subset of $\log_2(N+1)$ random features is subset (N) Total feature.

   Other methods of choosing are:
   - Sqrt (N)
   - $N/3$

3. Bootstrap samples: Allowing to choose same row more than once while training DT subset.
   - It affects the overall result of DT.

4. OOB error: we fit the rows that are not considered by bootstraping during training, and get the $\hat{y}$ and calculate the OOB error.
   - OOB doesn't affect the model.

⇒ code:

```
1. From sklearn.ensemble import Rando Forest Classifier.
2. model = Random Forest Classifier (n_estimator = 10,
                                     max_feature = 'auto',
                                     random state = 101)
3. model.fit (x_train, y_train)
4. model_pred = model.predict (x_test)
5. Evaluate.
6. Perform Grid search cv to choose best Hyper parameter
7. # choosing correct no. of trees
   test err = []
   For n in range (1,401)
      model = Random forest Classifier (n.estimators = n,
                                        max_features = 'auto')
      model.fit (x_train, y_train)
      test_pred = model.predict (x_test)
      test_error_append (1 - accuracy_score (test_pred, y_test))

plt.plot (range(1,40), test_error, label = 'Testerror')
plt.legend()
```