COSC 3337 : Data Science I

N. Rizk

College of Natural and Applied Sciences

Department of Computer Science

University of Houston

Metrics To Evaluate Machine Learning Algorithms in Python

Choice of metrics influences how the performance of machine learning algorithms is measured and compared.

They influence how you weight the importance of different characteristics in the results and your ultimate choice of which algorithm to choose.

Metrics are demonstrated for both **classification** and **regression** type machine learning problems.

•For classification metrics, the Pima Indians onset of diabetes dataset is used as demonstration. This is a binary classification problem where all of the input variables are numeric

•For regression metrics, the Boston House Price dataset is used as demonstration. this is a regression problem where all of the input variables are numeric

Classification Metrics

1. Classification Accuracy.
2. Logarithmic Loss.
3. Area Under ROC Curve.
4. Confusion Matrix.
5. Classification Report.

The cross_val_score() function simply computes the R2 for each fold of the cross validation, in case of regression models

cross_val_score is implemented in sklearn.model_selection and not in the base sklearn package itself.

```
In [77]: xtr, xts, ytr, yts = train_test_split(X, y, test_size = 0.2)

In [78]: regr_model = LinearRegression(n_jobs=-1)
         regr_model.fit(xtr, ytr)

Out[78]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=-1, normalize=False)
```

Cross validation using cross_val_score for five-fold CV

```
In [88]: cv_scores = cross_val_score(regr_model, X, y, cv= 5)
         print("CV scores for each fold: ", cv_scores , '\nMean of CV scores: ',
               np.mean(cv_scores))

CV scores for each fold:  [ 0.97467604  0.97751403  0.96845209  0.97719745  0.97445288]
Mean of CV scores:  0.974458498631
```

Cross validation using R^2 from sklearn.metrics

```
In [89]: from sklearn.metrics import r2_score
         predictions = regr_model.predict(xts)
         print ("R^2 score for holding CV: ", r2_score(yts, predictions))

R^2 score for holding CV:  0.972829847473
```

1.Classification Accuracy

Classification accuracy is the number of correct predictions made as a ratio of all predictions made.

```
# Cross Validation Classification Accuracy
import pandas
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
url =
"https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima
-indians-diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi',
'age', 'class']
dataframe = pandas.read_csv(url, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
seed = 7
kfold = model_selection.KFold(n_splits=10, random_state=seed)
model = LogisticRegression()
scoring = 'accuracy'
results = model_selection.cross_val_score(model, X, Y, cv=kfold,
scoring=scoring)
print("Accuracy: %.3f (%.3f)") % (results.mean(), results.std())
```

Accuracy: 0.770 (0.048)

2.Logarithmic Loss.

Logarithmic loss (or logloss) is a performance metric for evaluating the predictions of probabilities of membership to a given class.

Log-loss is a measurement of accuracy that incorporates the idea of probabilistic confidence given by following expression for binary class:

$$-(y\log(p)+(1-y)\log(1-p))$$

It takes into account the uncertainty of your prediction based on how much it varies from the actual label. In the worst case, let's say you predicted 0.5 for all the observations. So log-loss will become -log(0.5) = 0.69. Hence, we can say that anything above 0.6 is a very poor model considering the actual probabilities.

2.Logarithmic Loss.

```
# Cross Validation Classification LogLoss
import pandas
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = pandas.read_csv(url, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
seed = 7
kfold = model_selection.KFold(n_splits=10, random_state=seed)
model = LogisticRegression()
scoring = 'neg_log_loss'
results = model_selection.cross_val_score(model, X, Y, cv=kfold, scoring=scoring)
print("Logloss: %.3f (%.3f)") % (results.mean(), results.std())
```

Logloss: -0.493 (0.047)

3. Area Under ROC Curve. (Receiver operating characteristic)

Area under ROC Curve (or AUC for short) is a performance metric for **binary classification** problems.

The AUC represents a model's ability to discriminate between positive and negative classes. An area of 1.0 represents a model that made all predictions perfectly. An area of 0.5 represents a model as good as random.

ROC can be broken down into sensitivity and specificity. A binary classification problem is really a trade-off between sensitivity and specificity.

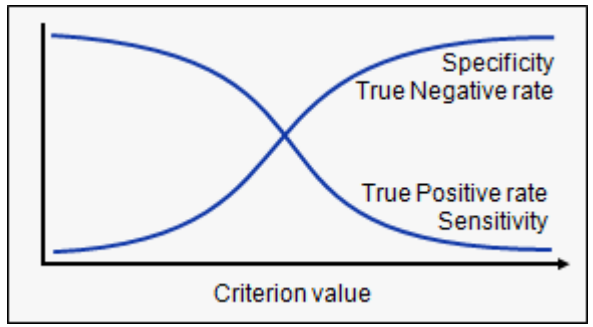
Sensitivity is the true positive rate also called the recall. It is the number instances from the positive (first) class that actually predicted correctly.

Specificity is also called the true negative rate. Is the number of instances from the negative class (second) class that were actually predicted correctly.

Confusion Matrix		Target			
		Positive	Negative		
Model	Positive	a	b	*Positive Predictive Value*	a/(a+b)
	Negative	c	d	*Negative Predictive Value*	d/(c+d)
		Sensitivity	*Specificity*	**Accuracy** = (a+d)/(a+b+c+d)	
		a/(a+c)	d/(b+d)		

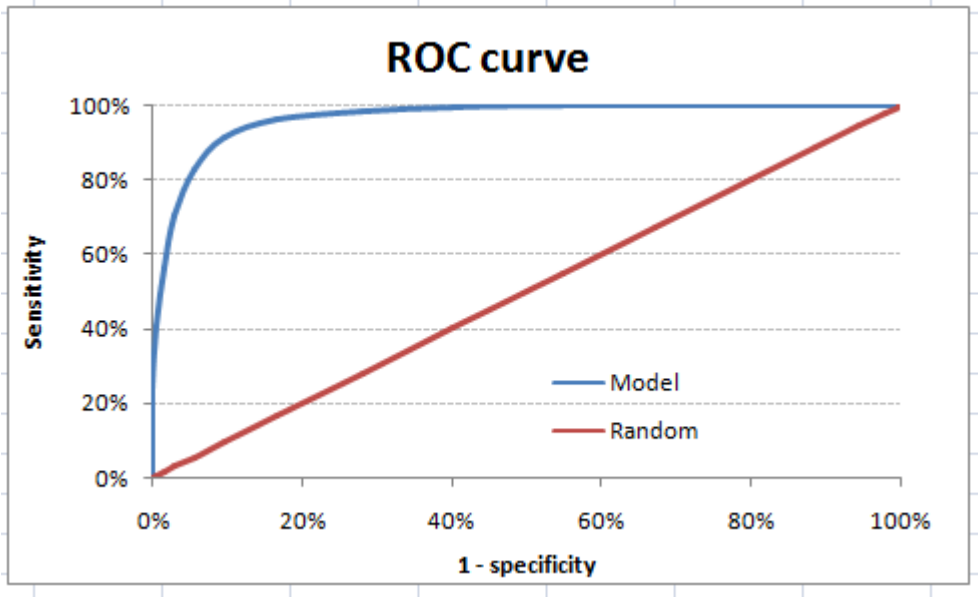
Advantage: ROC curve is almost independent of the response rate.

The ROC curve is the plot between sensitivity and (1- specificity). (1- specificity) is also known as false positive rate and sensitivity is also known as True Positive rate



With threshold 0.5

Count of ID	Target			
Model	1	0	Grand Total	
1	3,834	639	4,473	85.7%
0	16	951	967	1.7%
Grand Total	3,850	1,590	5,440	
	99.6%	40.19%		88.0%



The sensitivity is 99.6% and the (1-specificity) is ~60%. This coordinate becomes on point in our ROC curve. To bring this curve down to a single number, we find the area under this curve (AUC).

```
import numpy as np
from sklearn import metrics
y = np.array([1, 1, 2, 2])
pred = np.array([0.1, 0.4, 0.35, 0.8])
fpr, tpr, thresholds = metrics.roc_curve(y,
pred, pos_label=2)
metrics.auc(fpr, tpr)
```

0.75

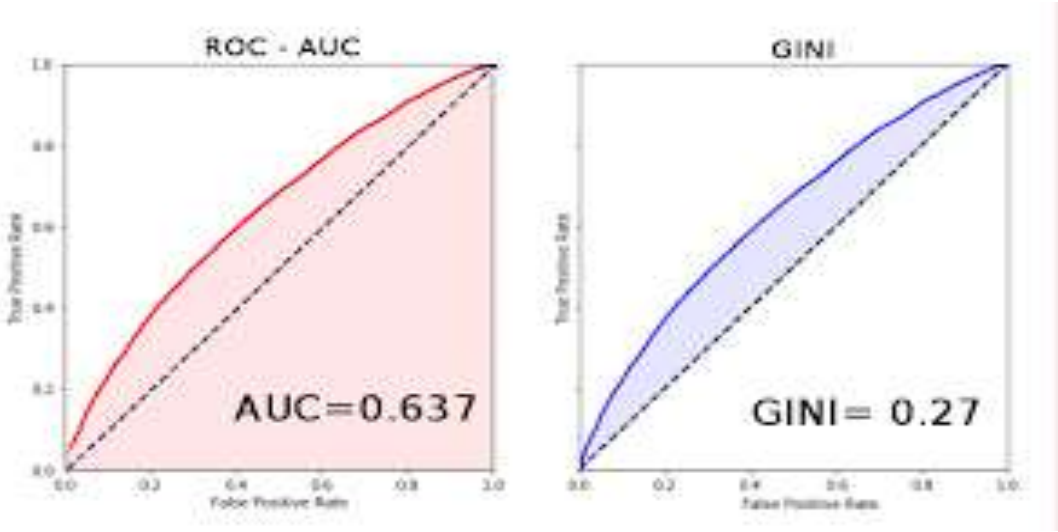
- .90-1 = excellent (A)
- .80-.90 = good (B)
- .70-.80 = fair (C)
- .60-.70 = poor (D)
- .50-.60 = fail (F)

Gini Coefficient

- Gini coefficient is sometimes used in **classification** problems. Gini coefficient can be straight away derived from the AUC ROC number.
- AUC itself is the ratio under the curve and the total area
- Gini is nothing but ratio between area between the ROC curve and the diagonal line & the area of the above triangle. Following is the formulae used :

- Gini = 2*AUC – 1

- Gini above 60% is a good model.



Cross Validation Classification ROC AUC

```
import pandas
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
url =
"https://raw.githubusercontent.com/jbrownlee/Datasets/master
/pima-indians-diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass',
'pedi', 'age', 'class']
dataframe = pandas.read_csv(url, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
seed = 7
kfold = model_selection.KFold(n_splits=10,
random_state=seed)
model = LogisticRegression()
scoring = 'roc_auc'
results = model_selection.cross_val_score(model, X, Y,
cv=kfold, scoring=scoring)
print("AUC: %.3f (%.3f)") % (results.mean(), results.std())
```

4. Confusion Matrix..

A confusion matrix is an N X N matrix, where N is the number of classes being predicted.

Here are a few definitions, you need to remember for a confusion matrix :

Accuracy : the proportion of the total number of predictions that were correct.

Positive Predictive Value or Precision : the proportion of positive cases that were correctly identified.

Negative Predictive Value : the proportion of negative cases that were correctly identified.

Sensitivity or Recall : the proportion of actual positive cases which are correctly identified.

Specificity : the proportion of actual negative cases which are correctly identified.

Confusion Matrix		Target			
		Positive	Negative		
Model	Positive	a	b	*Positive Predictive Value*	a/(a+b)
	Negative	c	d	*Negative Predictive Value*	d/(c+d)
		Sensitivity	*Specificity*	**Accuracy** = (a+d)/(a+b+c+d)	
		a/(a+c)	d/(b+d)		

Confusion Matrix		Target			
		Positive	Negative		
Model	Positive	a	b	*Positive Predictive Value*	a/(a+b)
	Negative	c	d	*Negative Predictive Value*	d/(c+d)
		Sensitivity	*Specificity*	**Accuracy** = (a+d)/(a+b+c+d)	
		a/(a+c)	d/(b+d)		

Count of ID	Target			
Model	1	0	Grand Total	
1	3,834	639	4,473	85.7%
0	16	951	967	1.7%
Grand Total	3,850	1,590	5,440	
	99.6%	40.19%		88.0%

4. Confusion Matrix..

```
# Cross Validation Classification Confusion Matrix
import pandas
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-
indians-diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age',
'class']
dataframe = pandas.read_csv(url, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
test_size = 0.33
seed = 7
X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,
test_size=test_size, random_state=seed)
model = LogisticRegression()
model.fit(X_train, Y_train)
predicted = model.predict(X_test)
matrix = confusion_matrix(Y_test, predicted)
print(matrix)
```

```
[[141  21]
 [ 41  51]]
```

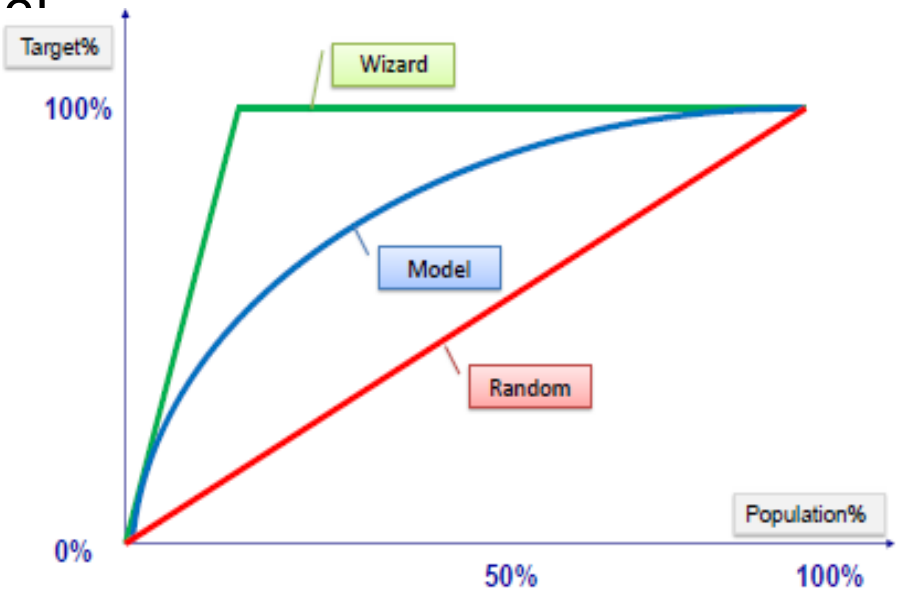
5. Classification Report.

```
# Cross Validation Classification Report
import pandas
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-
diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = pandas.read_csv(url, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
test_size = 0.33
seed = 7
X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,
test_size=test_size, random_state=seed)
model = LogisticRegression()
model.fit(X_train, Y_train)
predicted = model.predict(X_test)
report = classification_report(Y_test, predicted)
print(report)
```

	precision	recall	f1-score	support
0.0	0.77	0.87	0.82	162
1.0	0.71	0.55	0.62	92
avg / total	0.75	0.76	0.75	254

Gain and Lift charts

Lift is a measure of the effectiveness of a predictive mod
calculated as the ratio between the results obtained with
without the predictive model.
Cumulative gains and lift charts are visual aids for measu
model performance
Both charts consist of a lift curve and a baseline
The greater the area between the lift curve
and the baseline, the better the model



A company wants to do a mail marketing campaign. It costs the company $1 for each item mailed. They have information on 100,000 customers. Create a cumulative gains and a lift chart from the following data.

Overall Response Rate: If we assume we have no model other than the prediction of the overall response rate, then we can predict the number of positive responses as a fraction of the total customers contacted. Suppose the response rate is 20%. If all 100,000 customers are contacted we will receive around 20,000 positive responses.

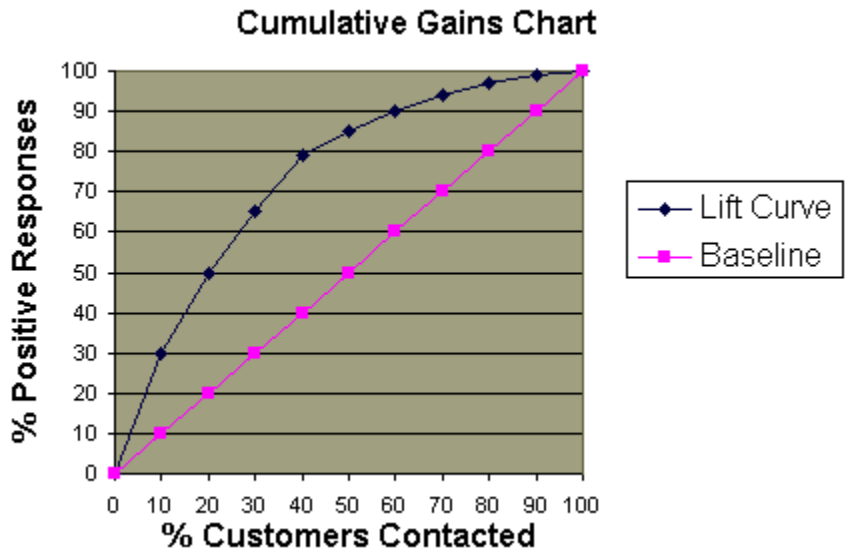
Cost ($)	Total Customers Contacted	Positive Responses
100000	100000	20000

Prediction of Response Model: A response model predicts who will respond to a marketing campaign. If we have a response model, we can make more detailed predictions. For example, we use the response model to assign a score to all 100,000 customers and predict the results of contacting only the top 10,000 customers, the top 20,000 customers, etc.

Cost ($)	Total Customers Contacted	Positive Responses
10000	10000	6000
20000	20000	10000
30000	30000	13000
40000	40000	15800
50000	50000	17000
60000	60000	18000
70000	70000	18800
80000	80000	19400
90000	90000	19800
100000	100000	20000

Cumulative Gains Chart:

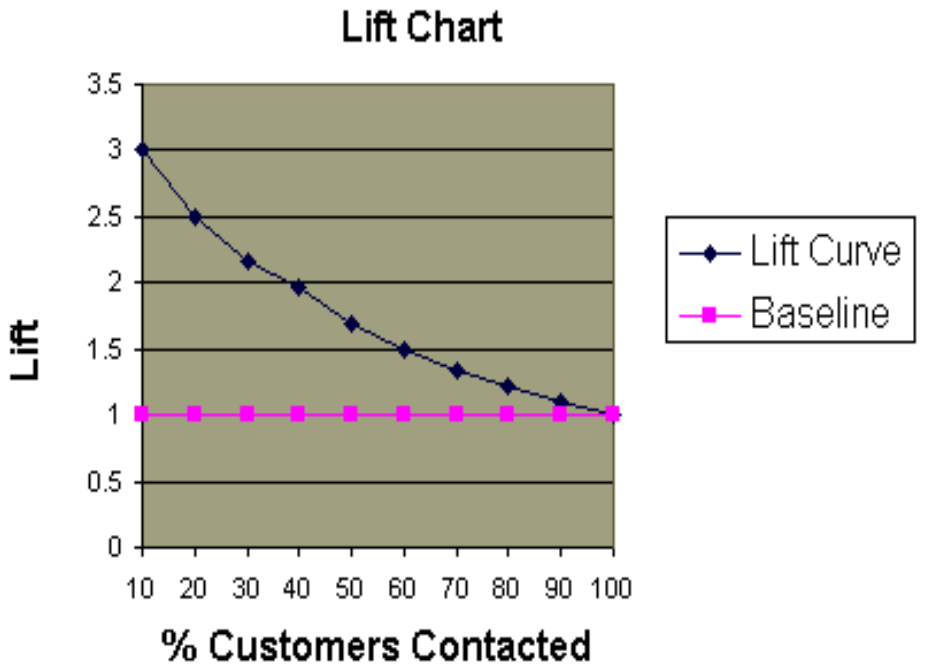
The y-axis shows the percentage of positive responses. This is a percentage of the total possible positive responses (20,000 as the overall response rate shows).
The x-axis shows the percentage of customers contacted, which is a fraction of the 100,000 total customers.
Baseline (overall response rate): If we contact X% of customers then we will receive X% of the total positive responses.
Lift Curve: Using the predictions of the response model, calculate the percentage of positive responses for the percent of customers contacted and map these points to create the lift curve.



Shows the actual lift.

To plot the chart: Calculate the points on the lift curve by determining the ratio between the result predicted by our model and the result using no model.

Example: For contacting 10% of customers, usin we should get 10% of responders and using the model we should get 30% of responders. The y- the lift curve at 10% is 30 / 10 = 3.



Evaluating a Predictive Model

We can assess the value of a predictive model by using the model to score a set of customers and then contacting them in this order. The actual response rates are recorded for each cutoff point, such as the first 10% contacted, the first 20% contacted, etc.

We create cumulative gains and lift charts using the actual response rates to see how much the predictive model would have helped in this situation. The information can be used to determine whether we should use this model or one similar to it in the future.

Optional

Applied Example

Using the response model P(x)=100-AGE(x) for customer xand the data table shown below, construct the cumulative gains and lift charts.

Customer Name	Height	Age	Actual Response
Alan	70	39	N
Bob	72	21	Y
Jessica	65	25	Y
Elizabeth	62	30	Y
Hilary	67	19	Y
Fred	69	48	N
Alex	65	12	Y
Margot	63	51	N
Sean	71	65	Y
Chris	73	42	N
Philip	75	20	Y
Catherine	70	23	N
Amy	69	13	N
Erin	68	35	Y
Trent	72	55	N
Preston	68	25	N
John	64	76	N
Nancy	64	24	Y
Kim	72	31	N
Laura	62	29	Y

Ties in ranking should be arbitrarily broken by assigning a higher rank to who appears first in the table.

1. **Calculate P(x) for each person x**
2. **Order the people according to rank P(x)**

Customer Name	P(x)	Actual Response
Alex	88	Y
Amy	87	N
Hilary	81	Y
Philip	80	Y
Bob	79	Y
Catherine	77	N
Nancy	76	Y
Jessica	75	Y
Preston	75	N
Laura	71	Y
Elizabeth	70	Y
Kim	69	N
Erin	65	Y
Alan	61	N
Chris	58	N
Fred	52	N
Margot	49	N
Trent	45	N
Sean	35	Y
John	24	N

3. Calculate the percentage of total responses for each cutoff point

- Response Rate = Number of Responses / Total Number of Responses (10 yes)

Customer Name	P(x)	Actual Response
Alex	88	Y
Amy	87	N
Hilary	81	Y
Philip	80	Y
Bob	79	Y
Catherine	77	N
Nancy	76	Y
Jessica	75	Y
Preston	75	N
Laura	71	Y
Elizabeth	70	Y
Kim	69	N
Erin	65	Y
Alan	61	N
Chris	58	N
Fred	52	N
Margot	49	N
Trent	45	N
Sean	35	Y
John	24	N

Total Customers Contacted	Number of Response	Responses Rate
2	1	10%
4	3	30%
6	4	40%
8	6	60%
10	7	70%
12	8	80%
14	9	90%
16	9	90%
18	9	90%
20	10	100%

4. Create the cumulative gains chart:

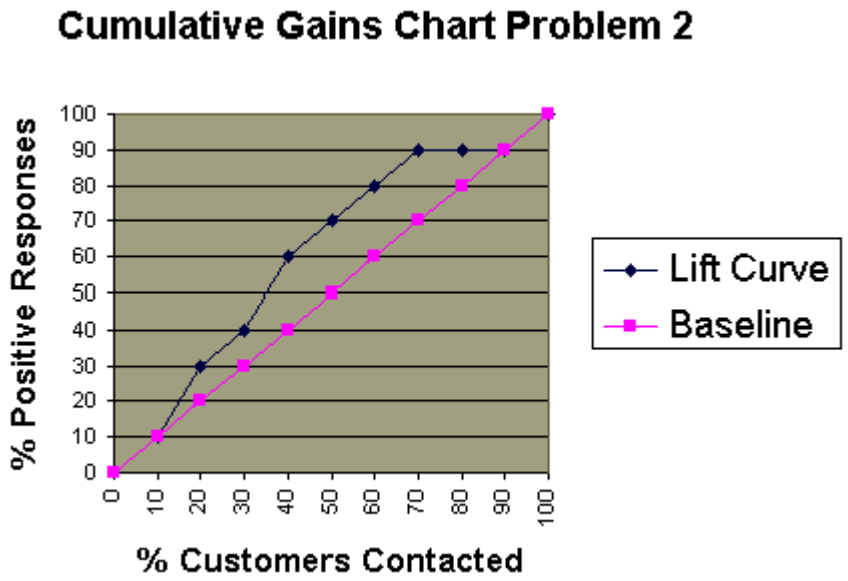
•The lift curve and the baseline have the same values for 10%-20% and 90%-100%.

The y-axis shows the percentage of positive responses. This is a percentage of the total possible positive responses (20,000 as the overall response rate shows).

The x-axis shows the percentage of customers contacted, which is a fraction of the 100,000 total customers.

Baseline (overall response rate): If we contact X% of customers then we will receive X% of the total positive responses.

Lift Curve: Using the predictions of the response model, calculate the percentage of positive responses for the percent of customers contacted and map these points to create the lift curve.

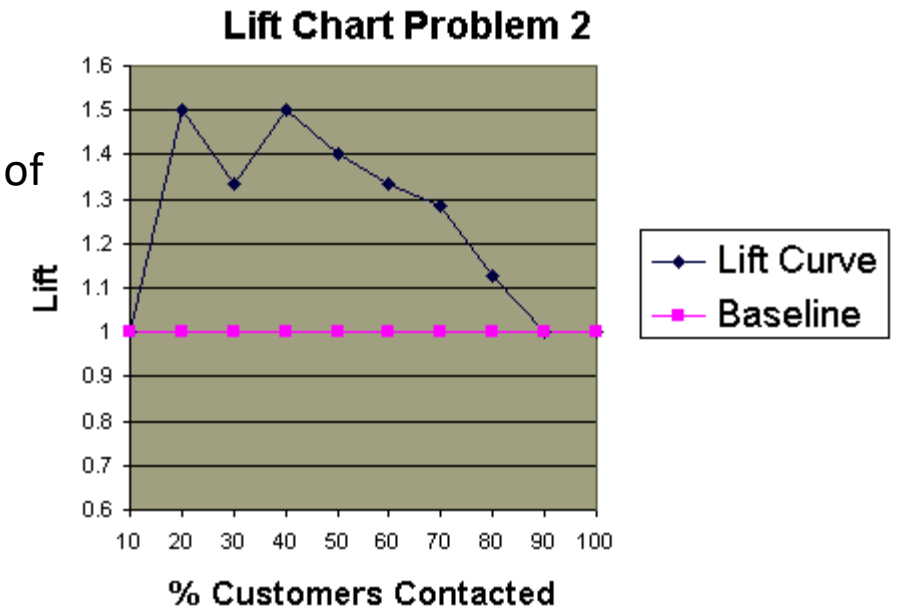


5. Create the lift chart:

Shows the actual lift.

To plot the chart: Calculate the points on the lift curve by determining the ratio between the result predicted by our model and the result using no model.

Example: For contacting 10% of customers, using no model we should get 10% of responders and using the given model we should get 30% of responders. The y-value of the lift curve at 10% is 30 / 10 = 3.



Regression Metrics

1. Mean Absolute Error.
2. Mean Squared Error.
3. R^2

1. Mean Absolute Error

The Mean Absolute Error (or MAE) is the sum of the absolute differences between predictions and actual values. It gives an idea of how wrong the predictions were.

```
# Cross Validation Regression MAE
import pandas
from sklearn import model_selection
from sklearn.linear_model import LinearRegression
url =
"https://raw.githubusercontent.com/jbrownlee/Datasets/master/hou
sing.data"
names = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE',
'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MEDV']
dataframe = pandas.read_csv(url, delim_whitespace=True,
names=names)
array = dataframe.values
X = array[:,0:13]
Y = array[:,13]
seed = 7
kfold = model_selection.KFold(n_splits=10, random_state=seed)
model = LinearRegression()
scoring = 'neg_mean_absolute_error'
results = model_selection.cross_val_score(model, X, Y, cv=kfold,
scoring=scoring)
print("MAE: %.3f (%.3f)") % (results.mean(), results.std())
```

MAE: -4.005 (2.084)

Root Mean Squared Error (RMSE)

RMSE is the most popular evaluation metric used in regression problems. It follows an assumption that error are unbiased and follow a normal distribution.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N} (Predicted_i - Actual_i)^2}{N}}$$

- RMSE is highly affected by outlier values. Hence, make sure you've removed outliers from your data set prior to using this metric.
- As compared to mean absolute error, RMSE gives higher weightage and punishes large errors.

2. Mean Squared Error

```
# Cross Validation Regression MSE
import pandas
from sklearn import model_selection
from sklearn.linear_model import LinearRegression
url =
"https://raw.githubusercontent.com/jbrownlee/Datasets/master/housi
ng.data"
names = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS',
'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MEDV']
dataframe = pandas.read_csv(url, delim_whitespace=True,
names=names)
array = dataframe.values
X = array[:,0:13]
Y = array[:,13]
seed = 7
kfold = model_selection.KFold(n_splits=10, random_state=seed)
model = LinearRegression()
scoring = 'neg_mean_squared_error'
results = model_selection.cross_val_score(model, X, Y, cv=kfold,
scoring=scoring)
print("MSE: %.3f (%.3f)") % (results.mean(), results.std())
```

The Mean Squared Error (or MSE) is much like the mean absolute error in that it provides a gross idea of the magnitude of error.

MSE: -34.705 (45.574)

3. R^2 Metric

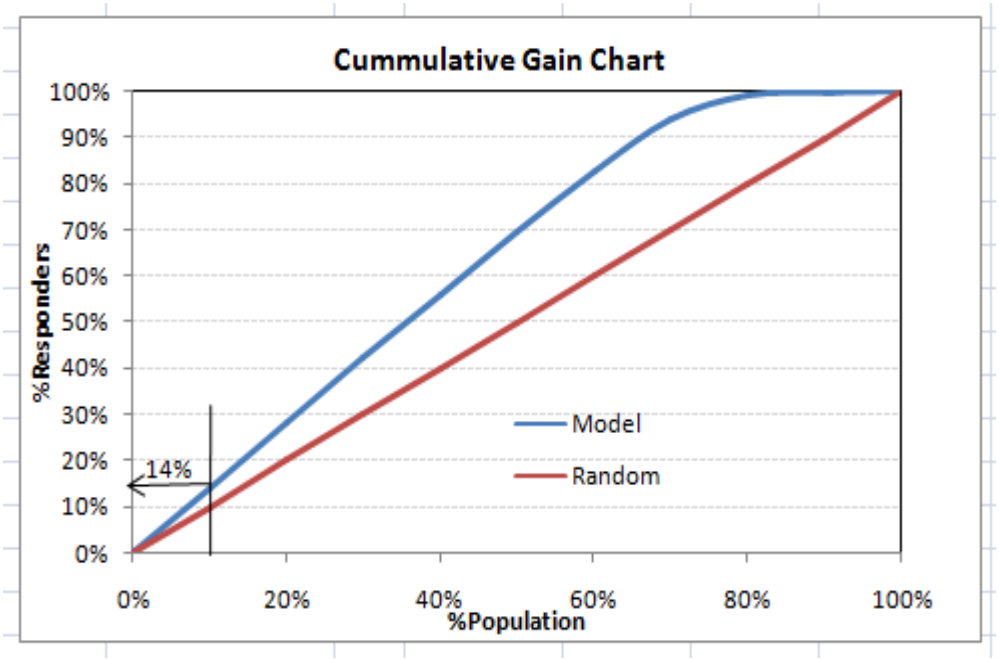
```
# Cross Validation Regression R^2
import pandas
from sklearn import model_selection
from sklearn.linear_model import LinearRegression
url =
"https://raw.githubusercontent.com/jbrownlee/Datasets/master/hous
ing.data"
names = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE',
'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MEDV']
dataframe = pandas.read_csv(url, delim_whitespace=True,
names=names)
array = dataframe.values
X = array[:,0:13]
Y = array[:,13]
seed = 7
kfold = model_selection.KFold(n_splits=10, random_state=seed)
model = LinearRegression()
scoring = 'r2'
results = model_selection.cross_val_score(model, X, Y, cv=kfold,
scoring=scoring)
print("R^2: %.3f (%.3f)") % (results.mean(), results.std())
```

The R^2 (or R Squared) metric provides an indication of the goodness of fit of a set of predictions to the actual values(coefficient of determination)

R^2: 0.203 (0.595)

This is a value between 0 and 1 for no-fit and perfect fit respectively.

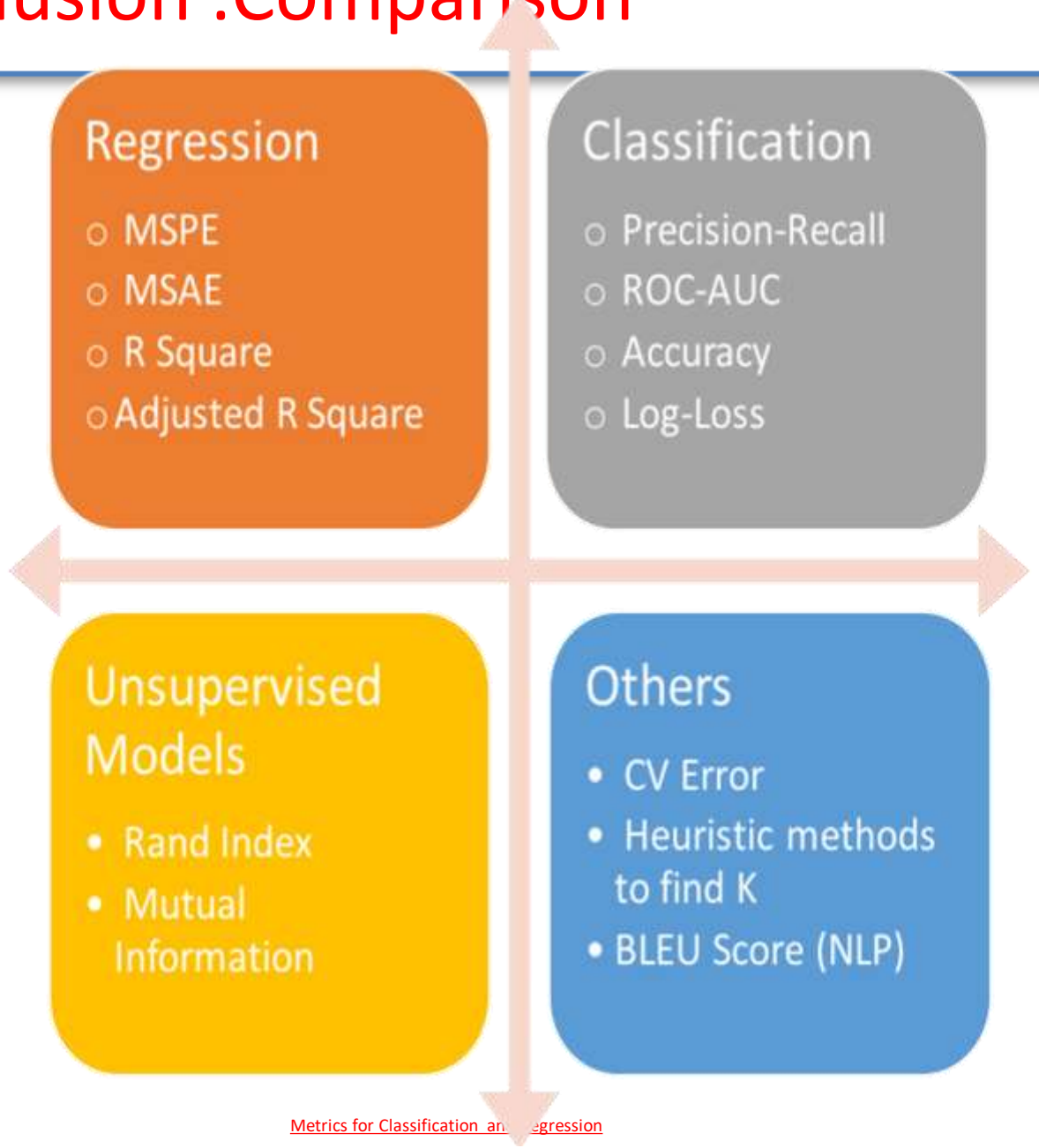
This is a very informative table. Cumulative Gain chart is the graph between Cumulative %Right and Cumulative %Population. For the case in hand here is the graph :

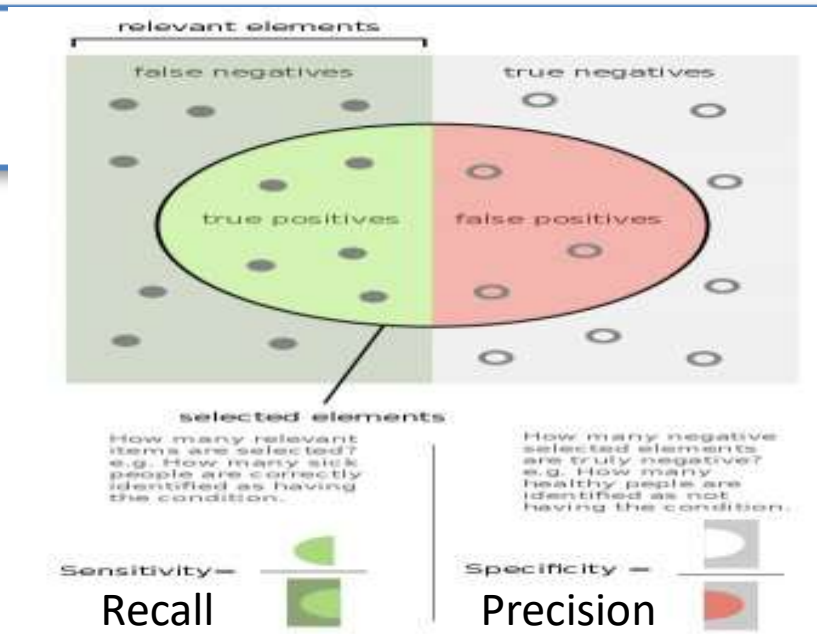


how well is the model segregating responders from non-responders. For example, the first decile however has 10% of the population, has 14% of responders. This means we have a 140% lift at first decile.

Lift / Gain charts are widely used in campaign targeting problems. This tells us till which decile can we target customers for an specific campaign. Also, it tells you how much response do you expect from the new target base.

Conclusion :Comparison





	Actual = Yes	Actual = No
Predicted = Yes	TP	FP Type Error I
Predicted = No	FN Type Error II	TN

- **F1 Score**: It is a harmonic mean of precision and recall given by-
F1 = 2*Precision*Recall/(Precision + Recall)
- **Accuracy:** Percentage of total items classified correctly- (TP+TN)/(N+P)

Log-loss is a measurement of accuracy that incorporates the idea of probabilistic confidence given by following expression for binary class:

Case 1 :Comparison of Log-loss with ROC & F1

S.No.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Actual (Balanced)	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
Predicted (Model 1)	0.1	0.1	0.1	0.1	0.1	0.1	0.6	0.6	0.5	0.5	0.9	0.9	0.9	0.9	0.9	0.9
Predicted (Model 2)	0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.7	0.7	0.7	0.7	0.8	0.8	0.8	0.8

Consider Case 1 (Balanced Data), it looks like model 1 is doing a better job in predicting the absolute probabilities whereas model 2 is working best in ranking observations according to their true labels. Let's verify with the actual score:

	F1 (threshold=0.5)	F1 (Threshold which maximize score)	ROC-AUC	Log-Loss
Model 1	0.88	0.88	0.94	0.28
Model 2	0.67	1	1	0.6

If you consider log-loss, Model 2 is worst giving a high value of log-loss because the absolute probabilities have big difference from actual labels. But this is in complete disagreement with F1 & AUC score, according to which Model 2 has 100% accuracy. Also, you would like to note that with different thresholds, F1 score is changing, and preferring model 1 over model 2 for default threshold of 0.5.

Case 2 How each of them deals with class imbalance?

S.No.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Actual (Imbalanced)	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
Predicted (Model 1)	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.9	0.9
Predicted (Model 2)	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.9	0.9	0.9	0.9

The only difference in the two models is their prediction for observation 13 & 14. Model 1 is doing a better job in classifying observation 13 (label 0) whereas Model 2 is doing better in classifying observation 14 (label 1). The goal is to see which model actually captures the difference in classifying the imbalanced class better (class with few observations, here it is label 1). In problems like fraud detection/spam mail detection, where positive labels are few, we would like our model to predict positive classes correctly and hence we will sometime prefer those model who are able to classify these positive labels

	F1 (threshold=0.5)	ROC-AUC	Log-Loss
Model 1	0.8	0.83	0.24
Model 2	0.86	0.96	0.24

Both F1 score and ROC-AUC score is doing better in preferring model 2 over model 1. So we can use both these methods for class imbalance.

But we will have to dig further to see how differently they treat class

S.No.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Actual (Imbalanced – few positive)	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
Predicted (Model 1)	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.9	0.9
Predicted (Model 2)	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.9	0.9	0.9	0.9

In the previous example, few positive labels. In the second example, there were few negative labels

Actual (Imbalanced – few negative)	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1
Predicted (Model 3)	0.1	0.1	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9
Predicted (Model 4)	0.1	0.1	0.1	0.1	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9

	F1 (threshold=0.5)	ROC-AUC	Log-Loss
Model 1	0.8	0.83	0.24
Model 2	0.86	0.96	0.24
Model 3	0.963	0.83	0.24
Model 4	0. 96	0.96	0.24

We can see that model (1) predicts 5 positives out of 100 true positives in a dataset of size 10K observations, while another model (2) predicts 90 positives out of 100 true positives. Clearly, model (2) is doing a much better job than model (1) in this case. Let's see if both F1 score & ROC-AUC score are able to capture that difference

F1 score for model (1) = 2*(1)*(0.1)/1.1 = 0.095
F1 score for model (2) = 2*(1)*(0.9)/1.9 = 0.947
Yes, the difference in F1 score reflects the model performance.

ROC-AUC for model (1) = 0.5
ROC-AUC for model (2) = 0.93

ROC-AUC gives a decent score to model 1 as well which is nota good indicator of its performance. Hence we should be careful while picking roc-auc for imbalanced datasets.

Table 3

Measures for multi-class classification based on a generalization of the measures of Table 1 for many classes C_i: tp_i are true positive for C_i, and fp_i – false positive, fn_i – false negative, and tn_i – true negative counts respectively. μ and M indices represent micro- and macro-averaging.

Measure	Formula	Evaluation focus
Average Accuracy	$\frac{\sum_{i=1}^{l}\frac{tp_i+tn_i}{tp_i+fn_i+fp_i+tn_i}}{l}$	The average per-class effectiveness of a classifier
Error Rate	$\frac{\sum_{i=1}^{l}\frac{fp_i+fn_i}{tp_i+fn_i+fp_i+tn_i}}{l}$	The average per-class classification error
$Precision_\mu$	$\frac{\sum_{i=1}^{l}tp_i}{\sum_{i=1}^{l}(tp_i+fp_i)}$	Agreement of the data class labels with those o
$Recall_\mu$	$\frac{\sum_{i=1}^{l}tp_i}{\sum_{i=1}^{l}(tp_i+fn_i)}$	Effectiveness of a classifier to identify class labe
$Fscore_\mu$	$\frac{(\beta^2+1)Precision_\mu Recall_\mu}{\beta^2 Precision_\mu+Recall_\mu}$	Relations between data's positive labels and thc
$Precision_M$	$\frac{\sum_{i=1}^{l}\frac{tp_i}{tp_i+fp_i}}{l}$	An average per-class agreement of the data class labels with those of a classifiers
$Recall_M$	$\frac{\sum_{i=1}^{l}\frac{tp_i}{tp_i+fn_i}}{l}$	An average per-class effectiveness of a classifier to identify class labels
$Fscore_M$	$\frac{(\beta^2+1)Precision_M Recall_M}{\beta^2 Precision_M+Recall_M}$	Relations between data's positive labels and those given by a classifier based on a per-class average

$$logloss = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij}\log(p_{ij})$$

Where,

- N *No of Rows in Test set*
- M *No of Fault Delivery Classes*
- Y_{ij} *1 if observation belongs to Class j; else 0*
- P_{ij} *Predicted Probability that observation belong to Class j*