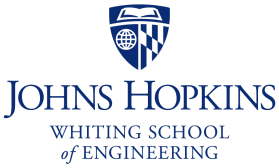
Model Evaluation



Introduction

Evaluating the machine learning model and inherently the development pipeline is an essential part of any ***learning*** project. There are two perspectives:

- Verifying and validating the learning approach/pipeline/model
- Deploying the ML model

Verifying the model always involves a training and testing dataset and frequently a validation dataset.

1. The training, testing, and validation datasets are drawn from the same main dataset. Thus, any underlying probabilistic distribution of the collected and preprocessed dataset is assumed to apply to all these three pieces.
2. Training dataset is used to develop the model. Training dataset is generally non-overlapping, or in other words, samples are drawn **without replacement**.
3. Validation dataset is used to **fine-tune** the model. Grid search model parameters and pick the best performance based on the **validation performance only**. Then the testing dataset is used to actually measure the **trained** and **tuned** model. If a model does not need a validation dataset (i.e. no parameters to tune, such as Naive Bayes), then we can add the validation dataset to the training dataset.
4. Testing dataset is used to verify and evaluate the model performance.

Important: Though evaluation is about measuring the performance, it is also setting the details of the model development pipeline in such a way that the exact pipeline will also be used for the **product** development or deployment.

Note that through model evaluation we are making sure that what we have done as model development is correct. If there is a next project step where we **deploy** the model in a computer system (in fact it would be our actual product), so that our developed model would actually predict something, then the most common approach is using the entire dataset to train and develop the model, exactly as the evaluation pipeline is constructed. Then a possible overall approach would be use train/test/validate to build and verify the pipeline, and compute some metrics so that we can show scholarly that the developed model actually learned something and performs better than a **random** classifier.

Deployment: Use entire dataset for training. The pipeline and the algorithm (and possibly multiple models) will make sure a robust (unbiased, not overfit, not underfit) model will be

trained. The evaluation assures that the overall method and the overall pipeline is sound.

Question: If there are 5 target classes, then what is the lower-bound accuracy a random classifier would achieve?

Model evaluation answers a few questions:

- Classification - What is the expected accuracy a model would achieve when given unseen data points and asked to predict?
- Classification - What is the expected accuracy of individual classes and confusion between them (i.e. when the class A is truth for a particular data point, is class B is predicted, or class C is predicted?)
- Regression - What is the total error between the regression curve (simplified model, fitted curve, etc.) and the actual data points?
- If we are OK with a particular **False-Alarm-Rate**, then what is the highest accuracy we can achieve? For example, based on Receiver Operating Characteristic (**ROC**) curve what should be our operating point.
- What is the variation of the classification error? (*aka* error rate of the model)
- Can we estimate the model generalization?

Model Evaluation Methods

Training-testing-validation: Common approach. In our project, we have to come up with a suitable training-testing-validation dataset. A good example can be as in the following: Given 20 year long dataset where we know the data is collected every day for 20 years, the training dataset can use all those 20 years. The validation dataset can use all 20 years but a percentage portion of the training (e.g. 10%). The testing dataset can be drawn from the last 5 years. Assuming the unseen data would be somewhat closer to the recent years. This kind of decisions are to be made by the model developer working with the subject-matter experts.

K-fold cross validation: Common approach. Parameter tuning is not done but parameters are fixed throughout the entire development and evaluation. Each fold has non-overlapping test and train datasets. Average the computed accuracies for each fold and report as the accuracy. Clearly, an ML method with **good generalization** and rather **insensitive to model parameters** would work much better with this evaluation model.

Leave-one-out-cross-validation (LOOCV): Seldom used. Similar to k-fold cross validation but testing dataset is always a single point. Recommended for very small datasets or when several outlier data points exist in the set, and they are to be *learned* by the model.

Stratification: When target categories are unbalanced (e.g. there are more benign cases rather than malignant), then a correct validation requires the training/test datasets, or folds, preserving the percentage of samples for each class. `sklearn` library supports stratification fully, such as by the use of `StratifiedKFold` library function.

The **most important aspect** of model evaluation is measuring the **expected** *real-world* performance with a good generalization.

Generalization

The developed model's ability to adapt (not a real adaptation but more like being ready) properly to new, previously unseen data, drawn from the *same probabilistic distribution* as the one used to create the model. Some classifiers which have good generalization ability:

- Support Vector Machines (large-margin) classifiers, since they solve an optimization problem to maximize the distance between data points and the decision (separating) surface.
- Random Forest (ensemble) classifiers, since they use few features per classifier, but numerous *simple* classifiers which randomly employ features, and then take the majority of the predictions made by this ensemble of classifiers.
- Principal Component Analysis (PCA) for unsupervised learning, since they try to find simplified data representation by looking at the largest eigenvalues of the covariance matrix of features.
- Human beings 😀

Evaluation Metrics

Following is a list of evaluation metrics.

1. Classification Accuracy

This the major metric we will use.

$$\text{Accuracy} = \frac{\texttt{number of correct predictions}}{\texttt{total number of predictions}}$$

2. Confusion Matrix

This is the second major metric, and it is defined for binary classification.

N=33	truth A	truth B
predicted A	10	3
predicted B	1	20

Interpretation in Statistics and Analogy

Historically from statistics, a positive and a negative class can be defined via a Null hypothesis (H_0), such as patient has cancer is `positive`, and does not have is `negative`. Assume class A is positive and class B is negative. Predicting A is accepting H_0 and predicting B is rejecting H_0. Then rejecting H_0 becomes a Type I error.

Semantically Type I errors are considered to be worse because they cause to conclude that a finding exists when in fact it does not, such as missing cancer, a wrongful death sentence, missing a nuclear attack, etc. Type I errors are very costly. Thus, the definition of H_0 by the statistical test designer has utmost importance. In machine learning we define categories according to the data and not following a statistical experiment.

Type I error - missing the truth, rejecting H_0, mis-predict class A - patient has cancer and missed.
Type II error - accepting H_0 when truth is False, patient does not have cancer but misdiagnosed - False Alarm.

True Positive (TP) = Predicted A, truth is A
True Negative (TN) = Predicted B, truth is B
False Positive (FP) = Predicted A, truth is B - Type II error
False Negative (FN) = Predicted B, truth is A - Type I error - major mistake

P_d - Probability of detection = TPR
P_f - Probability of false alarm = FPR

Example: TP=10, TN=20, FP=3, FN=1

Accuracy = $\frac{\texttt{TP}+\texttt{TN}}{\texttt{TP}+\texttt{TN}+\texttt{FP}+\texttt{FN}} = \frac{10+20}{10+20+3+1}$

TP Rate (TPR) = $\frac{\texttt{TP}}{\sum \texttt{truth positive}} = \frac{10}{10+1}$

TN Rate (TNR) = $\frac{\texttt{TN}}{\sum \texttt{truth negative}} = \frac{20}{20+3}$

Also, from Information Retrieval, following metrics are defined,

Precision = $\frac{\texttt{TP}}{\texttt{TP}+\texttt{FP}}$ How many selected items are relevant?

Recall = $\frac{\texttt{TP}}{\texttt{TP}+\texttt{FN}}$ How many relevant items are selected?

3. F1-score

Also known as harmonic mean of precision and recall.

F-1 score = $2\frac{\mathrm{Precision} \times \mathrm{Recall}}{\mathrm{Precision} + \mathrm{Recall}}$

F score attempts to find a balance between precision and recall

4. Area Under Curve (AUC)

Also known as area under Receiver Operating Characteristic (ROC) curve.

It is used for binary classification problem. AUC of a classifier is equal to the probability that the classifier will rank a randomly chosen positive example higher than a randomly chosen negative example. Note that in `sklearn` the `predict_proba()` is asusmed to generate real probabilities to compute an AUC. For some algorithms, such as NN or SVM, this assumption is not accurate.

5. Mean Absolute Error

(MAE) is the average error between the original x_i and the predicted $\hat{x}_i$ values for regression problems. Also known as L1 error.

MAE = $\frac{1}{N}\sum_{i=1}^{N}|x_i - \hat{x}_i|$

6. Mean Squared Error (MSE)

MSE is similar to MAE and is more common than MAE. Also known as L2 error.

MSE = $\frac{1}{N}\sum_{i=1}^{N}(x_i - \hat{x}_i)^2$

7. Logarithmic Loss

Also known as Log Loss measures false classifications and suitable for multi-class classification (rather less common error metric).

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

y_{ij} indicates whether sample i belongs to class j or not
p_{ij} indicates the probability (or score) of sample i belonging to class j

LogLoss has no upper bound, and it exists on the range $[0, \infty)$. Log Loss measure close to 0 indicates a higher accuracy. In general, minimizing LogLoss gives greater accuracy for the classifier.

Receiver Operating Characteristic Curve

The **ROC** curve is composed of true positive rate (**TPR**) on the y-axis and the false positive rate (**FPR**) on the x-axis while each operating point corresponds to some detection threshold or a classifier model parameter.

The best possible prediction method would yield a point in the upper left corner or coordinate (0,1) of the ROC space, representing zero false negatives and zero false positives, and perfect prediction. A random guess or a *coin flip* would result a point along the diagonal line (also called **line of no-separation**) from the left bottom to the top right corners (regardless of the positive and negative base rates).

ROC can be used for both model evaluation (such as the area under the ROC curve, presents the model behavior with different parameters) and, or deciding on a model parameter for deployment, i.e. setting the TPR with a given **accepted false alarm rate**. ROC is used to make a model selection or set an operating classifier threshold by considering the TPR-FPR together. Generally, ROC is used for binary classification.

An example ROC curve is generated in cells below using breast cancer data from `sklearn.datasets`.

In [1]:
```
%matplotlib inline
import matplotlib.pyplot as plt
plt.rcParams['figure.dpi'] = 72
import numpy as np
import pandas as pd
from sklearn.datasets import load_breast_cancer

# Locate and load the data file
bc = load_breast_cancer()
bc_df = pd.DataFrame(data= np.c_[bc.data, [bc.target_names[v] for v in bc.targe
                     columns= list(bc.feature_names)+['cancer'])

# See how the data looks like
bc_df.head()
```

Out[1]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry
0	17.99	10.38	122.8	1001.0	0.1184	0.2776	0.3001	0.1471	0.241
1	20.57	17.77	132.9	1326.0	0.08474	0.07864	0.0869	0.07017	0.181
2	19.69	21.25	130.0	1203.0	0.1096	0.1599	0.1974	0.1279	0.206
3	11.42	20.38	77.58	386.1	0.1425	0.2839	0.2414	0.1052	0.259
4	20.29	14.34	135.1	1297.0	0.1003	0.1328	0.198	0.1043	0.180

5 rows × 31 columns

In [2]:
```
# Populate the dataset, cancer column is target variable
X = bc_df.loc[:, bc_df.columns != 'cancer'].values
y = bc_df.loc[:, bc_df.columns == 'cancer'].values.ravel()
```

In [3]:
```
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
```

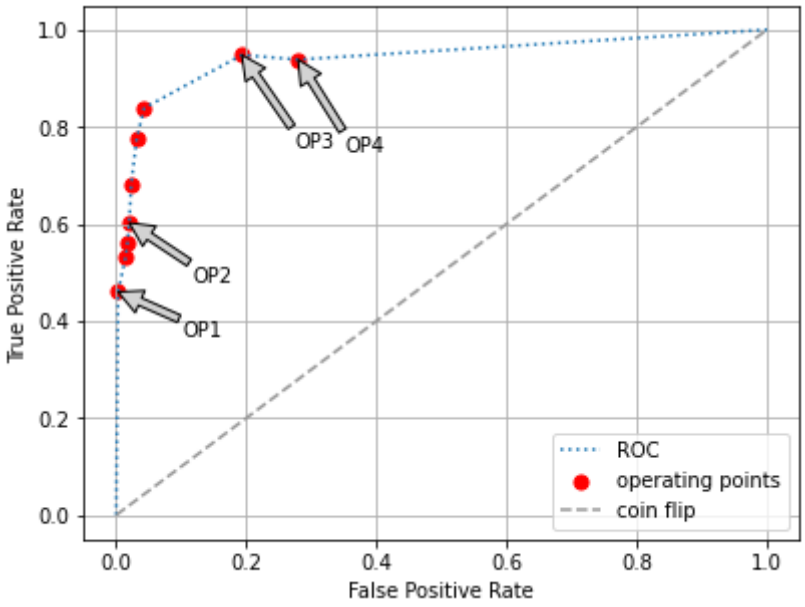
```
# Display OP
def annot(opi, _x, _y):
    plt.annotate(f"OP{opi}", xy=(_x, _y), xytext=(.90*_x+.1, .80*_y), arrowprop

# Training and testing datasets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.98, rando

# Parameter to vary for Logistic Regression
C = (2e-1, 0.5, 0.8, 1, 2, 5, 1e1, 2e1, 1e2)

# Let's vary C and generate training/testing sessions to collect data for ROC
print(f'{"Test Acc":>8s} {"C":>11s} {"TPR":>6s} {"FPR":>6s}')
FPR, TPR = [], []
for c in C:
    pipe_lr = make_pipeline(StandardScaler(),
                            LogisticRegression(random_state=14,
                                               penalty='l1',
                                               solver='liblinear',
                                               class_weight='balanced',
                                               C=c,
                                               multi_class='auto',
                                               max_iter=10000))
    pipe_lr.fit(X_train, y_train)
    y_pred = pipe_lr.predict(X_test)
    tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
    TPR += [tp/(tp+fn)]  # Pd
    FPR += [fp/(fp+tn)]  # Pf

    print(f'{pipe_lr.score(X_test, y_test):8.3f} {c:11.5f} {TPR[-1]:6.3f} {FPR[
```

```
Test Acc           C    TPR    FPR
   0.803     0.20000  0.938  0.280
   0.860     0.50000  0.948  0.193
   0.912     0.80000  0.839  0.043
   0.896     1.00000  0.777  0.032
   0.866     2.00000  0.682  0.023
   0.837     5.00000  0.602  0.020
   0.823    10.00000  0.559  0.017
   0.814    20.00000  0.531  0.014
   0.794   100.00000  0.460  0.003
```

In [4]:

```
# Sorts the points to display nicely on ROC
FPR, TPR = zip(*sorted(zip(FPR, TPR)))
fpr = [0.]+list(FPR)+[1.]; tpr = [0.]+list(TPR)+[1.]

# Plot
fig, ax = plt.subplots(dpi=72)
plt.plot(fpr, tpr, ':', label='ROC')
plt.scatter(FPR, TPR, 50, color='red', marker='o', label='operating points')
plt.plot([0, 1], [0, 1], linestyle='--', color=(0.6, 0.6, 0.6), label='coin fli

# Annotate certain operating points
annot(1, fpr[1], tpr[1])
annot(2, fpr[4], tpr[4])
annot(3, fpr[8], tpr[8])
annot(4, fpr[9], tpr[9])

# Labels
plt.xlabel('False Positive Rate')
```

```
plt.ylabel('True Positive Rate')
plt.legend(loc='lower right')
plt.grid()
plt.show()
```



Dramatization

Note that in order to generate a nice ROC for ***demonstration purposes***, we selected a harsh test size of 98% and `random_state=14`. Also picked `LogisticRegression` with parameters `penalty='l1'`, `solver='liblinear'`. Not every classifier generates an ROC curve like above due to non-existence of a parameter that can vary detection versus false alarm smoothly.

Important Question: Given above ROC, which operating point would you pick for cancer detection? OP1, OP2, OP3, or OP4?

Maximum a Posteriori (MAP) Estimation and ROC

In machine learning a probabilistic classification problem can be posed as a conditional probability with the two probability density functions (pdf) $p(x|H_0)$ and $p(x|H_1)$ for binary categories H_0 and H_1. Then a MAP model would attempt to place a detection threshold T_{MAP} on the variable x to predict H_0 or H_1. Such that if $x < T_{MAP}$ then the model decides (i.e. predicts) class label 0 else $x \geq T_{MAP}$ then the model decides (i.e. predicts) class label 1. Learning or fitting a model is equivalent to finding the right x value for the threshold T_{MAP}.

In Naive Bayes prediction model the prediction is based on this threshold, assuming the multidimensional variables x are independent, thus $p(X|H_k) = \prod_i p(x_i|H_k)$. This

independence assumption is part of Bayesian estimation and one more reason to do a feature selection in preprocessing.

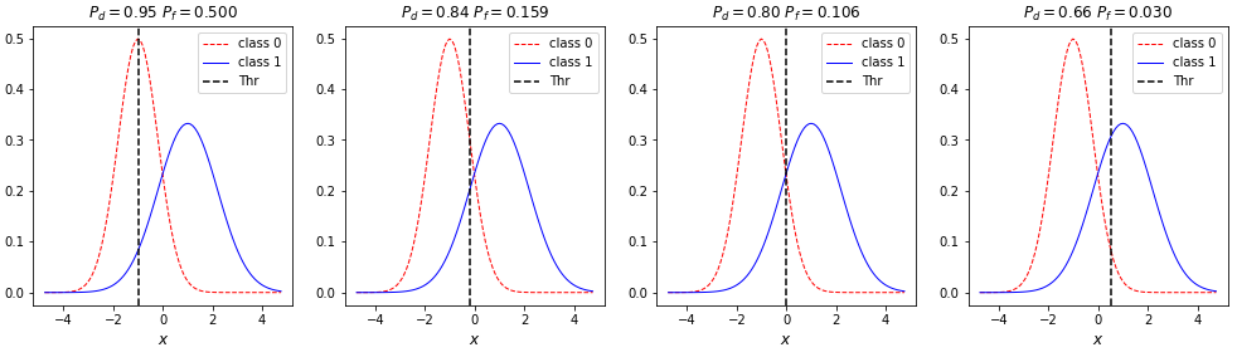
Since a probabilistic classification is based on probability density functions p, which can be derived from the training dataset, it is also natural to evaluate the model based on true positive and false positive rates achieved when a threshold is placed somewhere on the x axis. The ROC curve demonstrates this evaluation which is an optimization problem to find the best threshold with respect to true positive rate P_d and false positive rate P_f.

Below are two probability density functions where a detection is based on the class `0` pdf and class `1` pdf. The only parameter is the `Thr` where a prediction is made by comparing the new data point x to the Thr:

$$\text{prediction} = \begin{cases} \text{class } 0, & \text{if } x < Thr \\ \text{class } 1, & \text{if } x \geq Thr \end{cases}$$

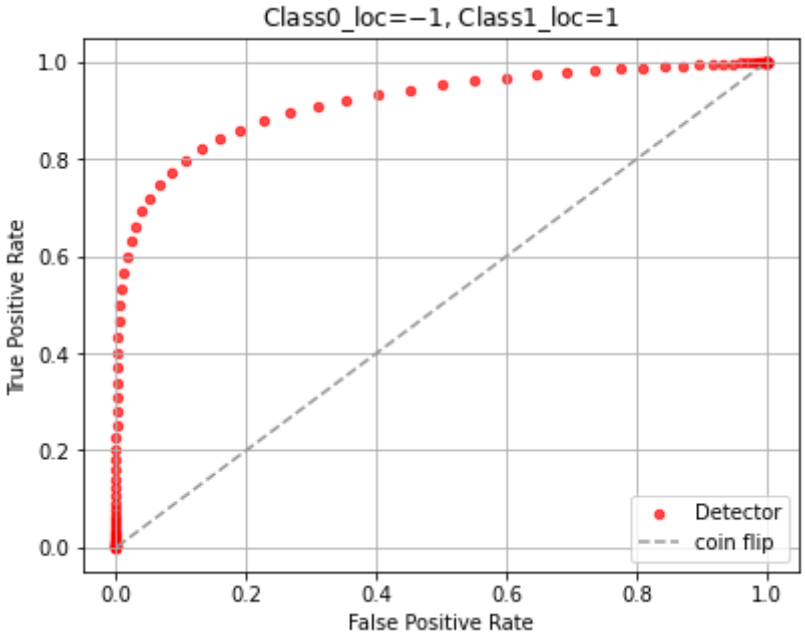
Consider two conditional pdf as below, class 0 is on the left and class 1 on the right. Compute the ROC based on numerous values of the T.

In [5]:
```
from scipy.stats import norm

# plot a normal distribution, loc=mean, scale=stdev
x = np.linspace(norm.ppf(0.000001), norm.ppf(0.999999), 100)  # 100 bins
norm0 = norm(loc=-1, scale=.8)
norm1 = norm(loc=1, scale=1.2)
pdf0 = lambda x : norm0.pdf(x)
pdf1 = lambda x : norm1.pdf(x)
cdf0 = lambda th : norm0.cdf(th)
cdf1 = lambda th : norm1.cdf(th)

def plot_det(_ax, _thr):
    _ax.plot(x, pdf0(x), 'r--', lw=1, label='class 0')
    _ax.plot(x, pdf1(x), 'b', lw=1, label='class 1')
    _ax.axvline(x=_thr, ls='--', color='k', label='Thr')
    _ax.set_xlabel('$x$', size='large')
    _ax.set_title(f'$P_d=${1-cdf1(_thr):.2f} $P_f=${1-cdf0(_thr):.3f}')
    _ax.legend()

plt.figure(figsize=(17, 4), dpi=72)

# different thresholds
plot_det(plt.subplot(1, 4, 1), -1)
plot_det(plt.subplot(1, 4, 2), -0.2)
plot_det(plt.subplot(1, 4, 3), 0)
plot_det(plt.subplot(1, 4, 4), 0.5)

plt.show()
```



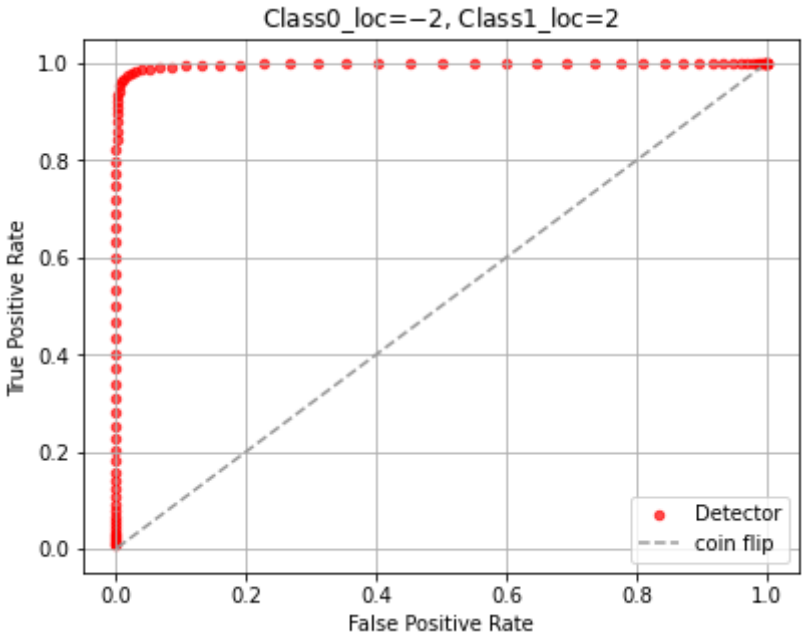
In [6]:
```
def plot_roc(_fpr, _tpr, _title):
    plt.scatter(_fpr, _tpr, s=20, marker='o', c='r', label='Detector', alpha=0.
    plt.plot(np.arange(0.001,1,0.01), np.arange(0.001,1,0.01), linestyle='--',
    plt.ylabel('True Positive Rate')
    plt.xlabel('False Positive Rate')
    plt.title(_title)
    plt.legend(loc='lower right')
    plt.grid()
    plt.show()
```

In [7]:
```
# Classical detector ROC
Tpr, Fpr = [], []
for thr in np.arange(-5, 5, .1):
    Tpr += [1-cdf1(thr)]
    Fpr += [1-cdf0(thr)]

plot_roc(Fpr, Tpr, 'Class0_loc=$-1$, Class1_loc=$1$')
```



In [8]:
```
# High separation, different pdf
cdf0 = lambda th : norm(loc=-2, scale=.8).cdf(th)
cdf1 = lambda th : norm(loc=2, scale=1.2).cdf(th)

Tpr, Fpr = [], []
for thr in np.arange(-5, 5, .1):
    Tpr += [1-cdf1(thr)]
    Fpr += [1-cdf0(thr)]
```

```
plot_roc(Fpr, Tpr, 'Class0_loc=$-2$, Class1_loc=$2$')
```



Generalization Error

From statistical learning theory, the **generalization error** is the difference between the expected and empirical error. Or, the difference between error on the training set and error on the underlying **joint** probability distribution.

Question: If we know the underlying joint probability distribution of the data, then would we need an ML model or method?

Answer: No. We will probably never be able to *know* the underlying data probability distribution for practical ML problems.

References

1. Frank, Eibe, et al. "Weka-a machine learning workbench for data mining." Data mining and knowledge discovery handbook. Springer, Boston, MA, 2009. 1269-1277.
2. Bousquet, Olivier, and André Elisseeff. "Stability and generalization." The Journal of Machine Learning Research 2 (2002): 499-526.
3. Bartlett, Peter. "For valid generalization the size of the weights is more important than the size of the network." Advances in neural information processing systems 9 (1996).

Exercises

Exercise 1. Switch the dataset to another one (either find one in `sklearn.datasets` or Kaggle) suitable for binary classification and plot the ROC curve for logistic regression classifier.

Exercise 2. Adjust the `loc` and `scale` for two conditional pdf above and turn the problem into a completely separable classification. Plot its ROC. What should be the AUC in this case?

In [9]:
```
%%html
<style>
    table {margin-left: 0 !important;}
</style>
<!-- Display markdown tables left oriented in this notebook. -->
```
