Cognixia

A Collabera LEARNING SOLUTIONS COMPANY

Additional Performance Evaluation and Model Selection

www.cognixia.com

Agenda

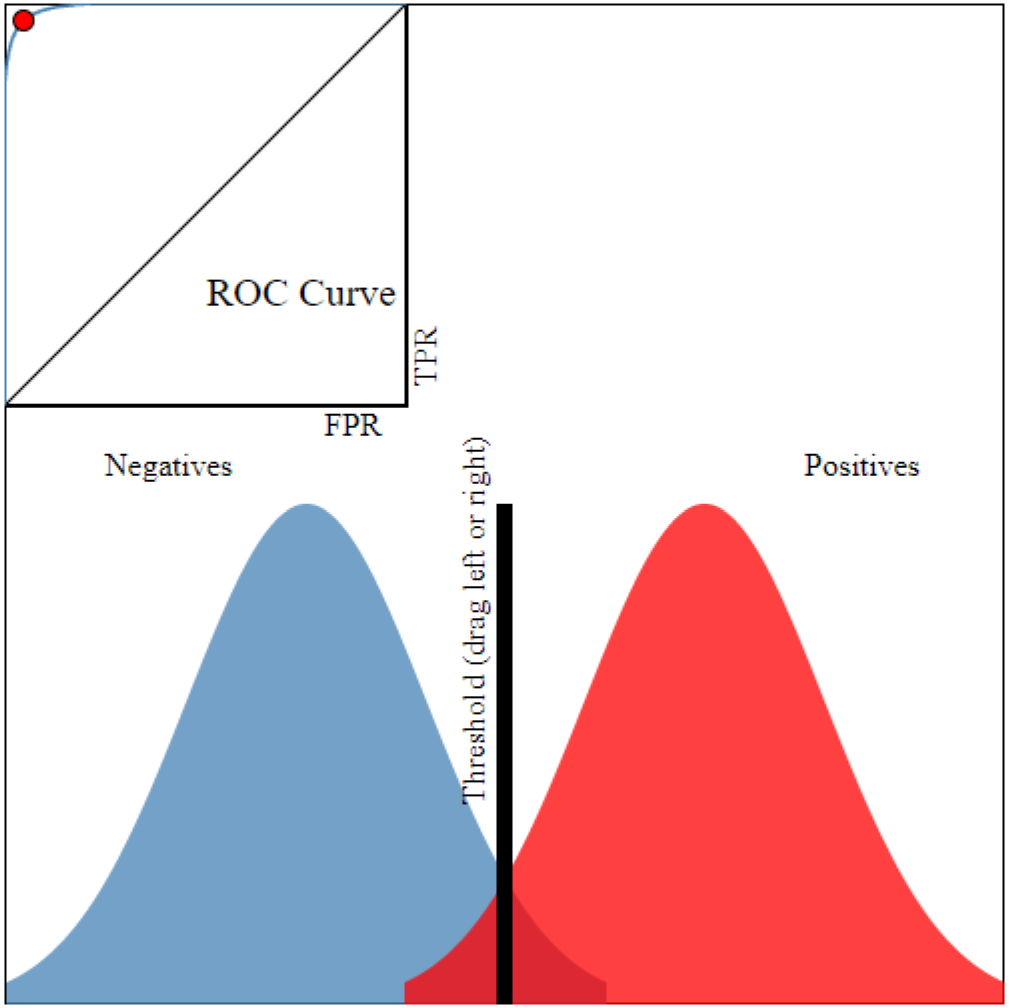
Performance Evaluation

Receiver Operating Characteristic (ROC)

- A receiver operating characteristic curve, ROC curve, is a graphical plot that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied. The ROC curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings.
- TPR: True Positive Rate corresponds to the scenario when the actual output is positive and our model has also predicted positive.
- FPR: False Positive Rate corresponds to the scenario when the actual output is negative and our model has predicted positive.
- Consider an example of the number of papers accepted into a scientific journal, the positive case is that the paper is admitted and the negative case is that the paper was not admitted.

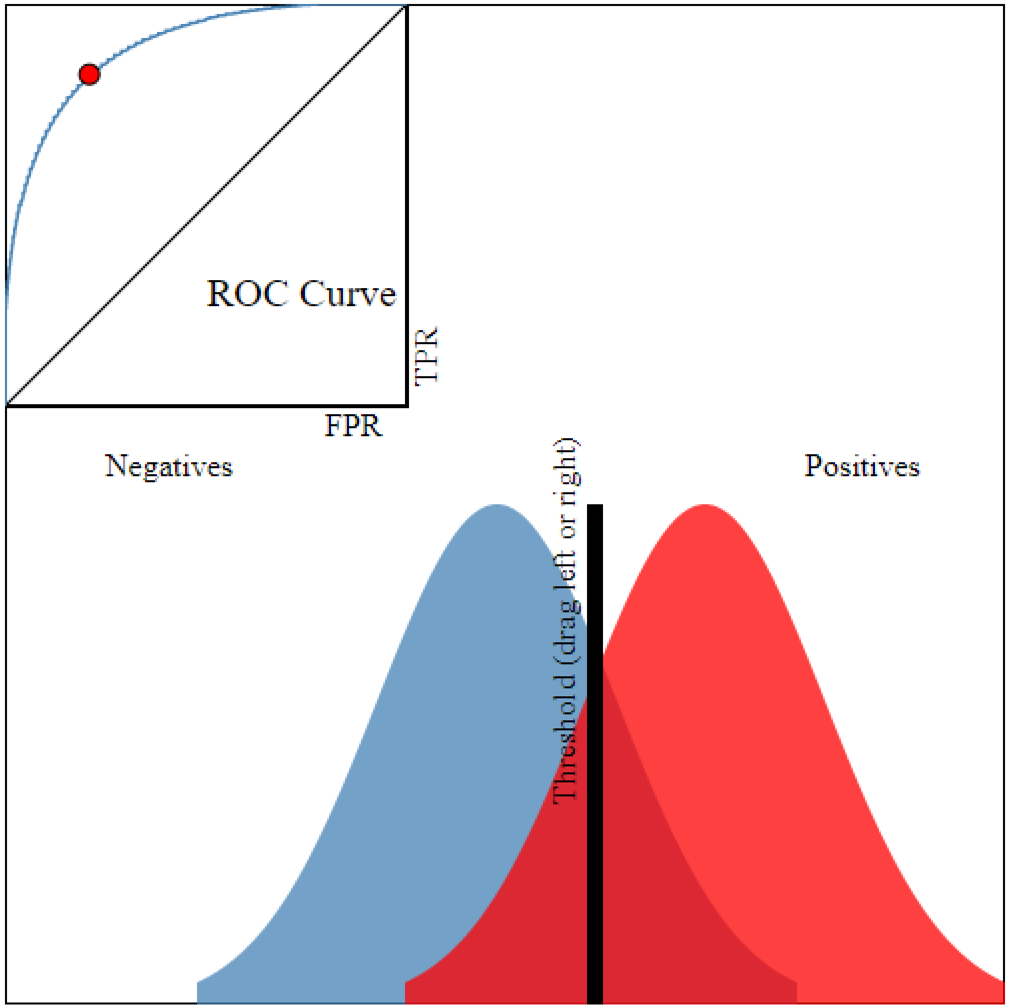
Example data set distribution

- Consider the blue curve representing the number of papers not admitted in the scientific journal and the red curve represents the number of papers admitted.
- Consider that each pixel in the curve is a paper.
- The black line represents the threshold for our model classifier. In this case the classification threshold is set to 50%.
- These Red and Blue curves represent the output of our binary classifier.



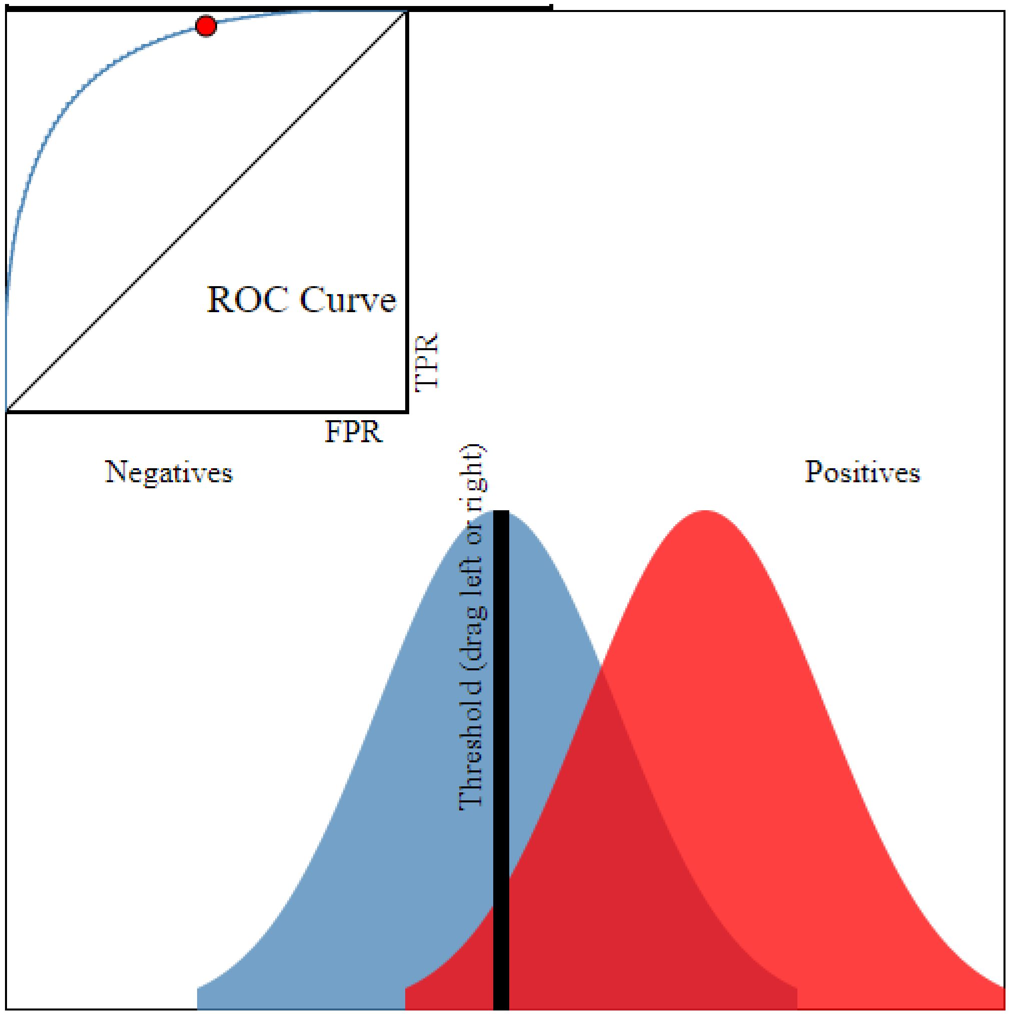
Example data set distribution

- Consider this case where our classifier has not predicted the results with good accuracy, as we can see a large overlap between the positive and the negative cases.
- The ROC cure here is the plot of the TPR vs FPR, for each of the threshold values from 0 to 1.
- The ROC curve helps us understand how well our classifier separates the two classes.
- The threshold where the ROC curve is near to its peak is considered a good threshold value for a classification model.



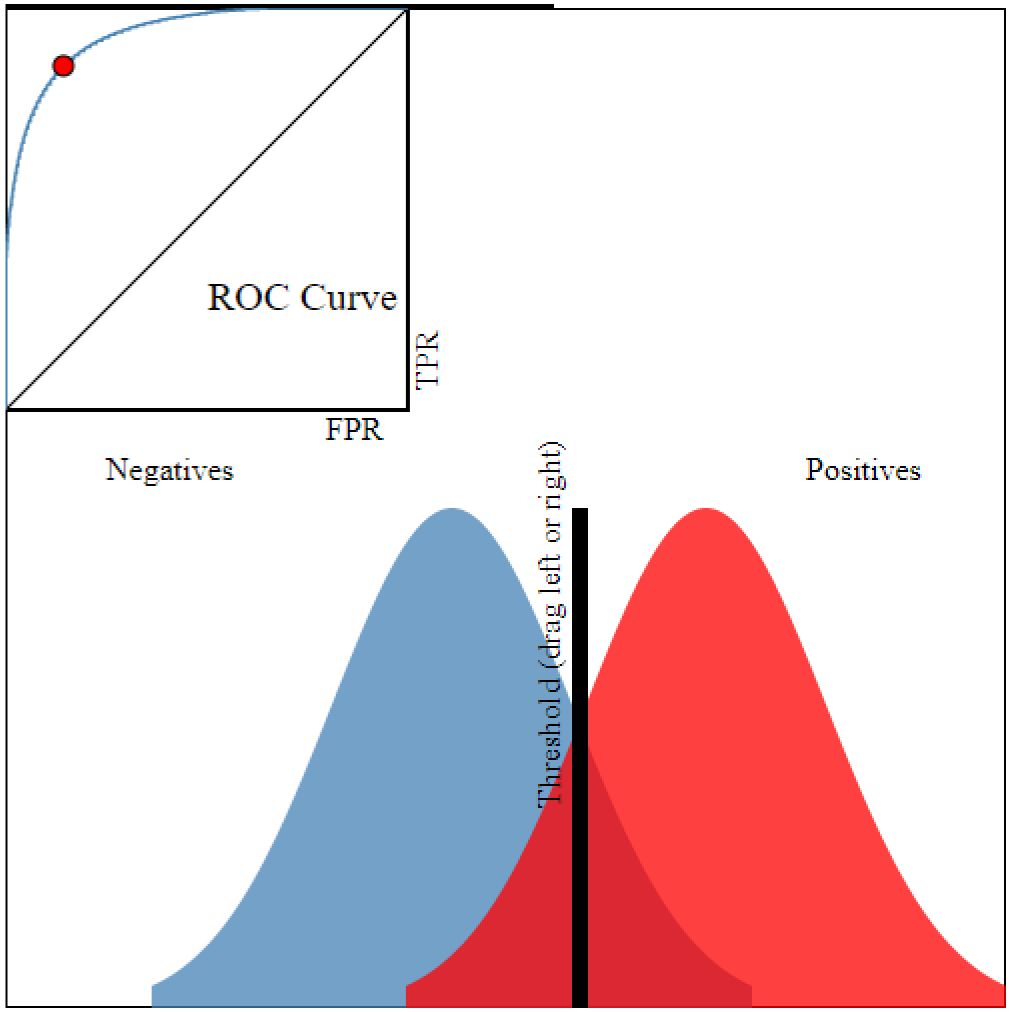
Example data set distribution

- Consider the case when the threshold is set lower than before.
- As we can see the ROC value went away from the peak, which tells us that this threshold for classification does not give a good separation between the two classes.



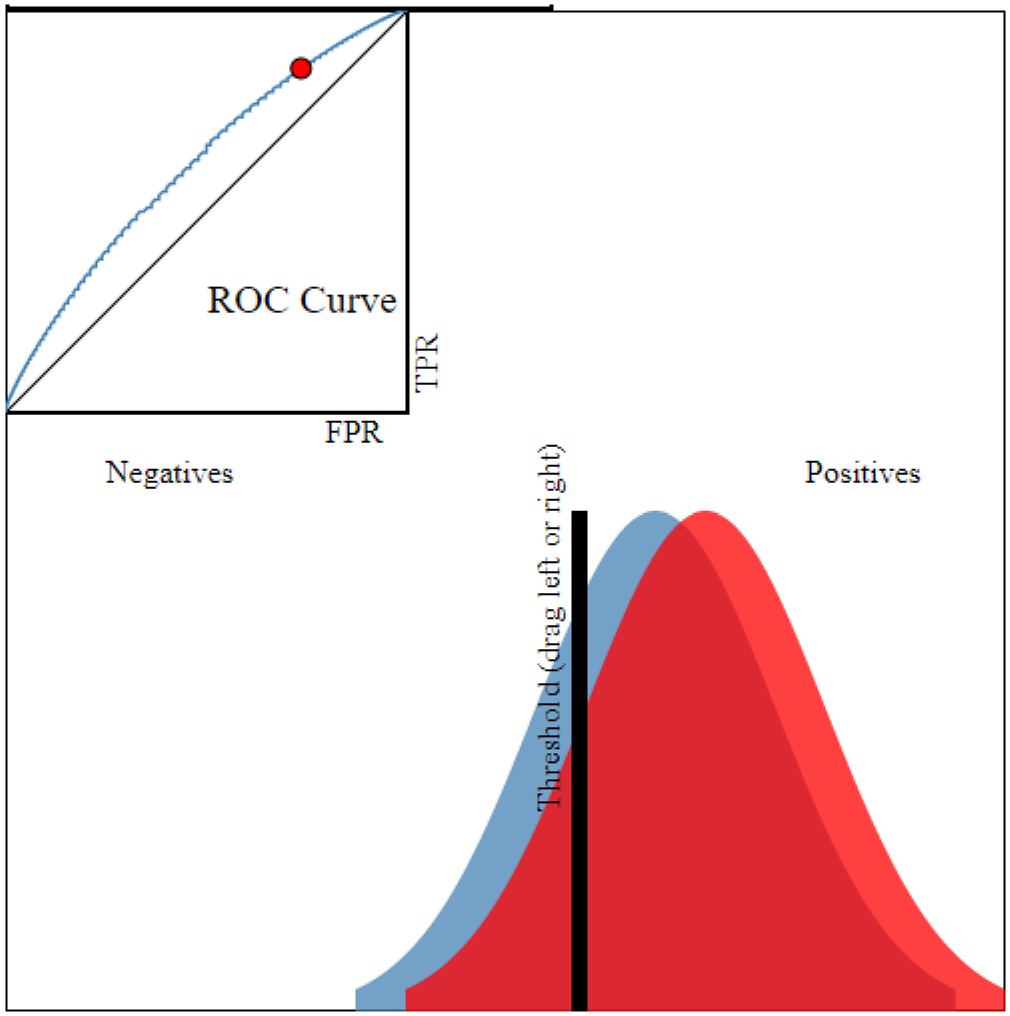
Area Under The Curve (AUC)

- Area under the curve, is the area under the ROC curve.
- The AUC is maximum when there is good separation between our predicted classification, i.e. our model has performed good classification.
- As we can see in this diagram the ROC curve almost cover the whole square.



Area Under The Curve (AUC)

- AUC is minimum, when our predicted values are not well separated, i.e. our model has given a bad classification result.
- As we can see in this diagram the ROC curve is reduced to covering only 50% for the area.



Silhouette Coefficient

- Silhouette refers to a method of interpretation and validation of consistency within clusters of data. The technique provides a succinct graphical representation of how well each object lies within its cluster.
- The silhouette value is a measure of how similar an object is to its own cluster (cohesion) compared to other clusters (separation). The silhouette ranges from −1 to +1, where a high value indicates that the object is well matched to its own cluster and poorly matched to neighbouring clusters.
- If most objects have a high value, then the clustering configuration is appropriate. If many points have a low or negative value, then the clustering configuration may have too many or too few clusters.
- Assume we have some clustered data, via some clustering technique like k-means.
- For each data point i, let a(i) be the average distance between i and all other data within the same cluster.

Silhouette Coefficient

- We can interpret a(i) as a measure of how well i is assigned to its cluster (the smaller the value, the better the assignment).
- We then define the average dissimilarity (difference / variance) of point i to a cluster c as the average of the distance from i to all points in c.
- Let b(i) be the smallest average distance of i to all points in any other cluster, of which i is not a member.
- The cluster with this smallest average dissimilarity (difference/variance) is said to be the "neighbouring cluster" of i because it is the next best fit cluster for point i.
- We now define a silhouette:

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}$$

Silhouette Coefficient

- Which can also be written as:

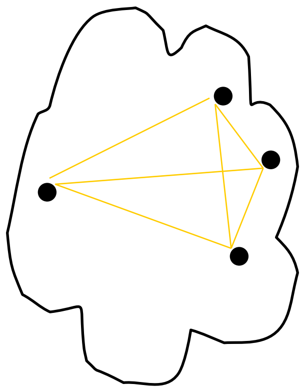
$$s(i) = \begin{cases} 1 - a(i)/b(i), & \text{if } a(i) < b(i) \\ 0, & \text{if } a(i) = b(i) \\ b(i)/a(i) - 1, & \text{if } a(i) > b(i) \end{cases}$$

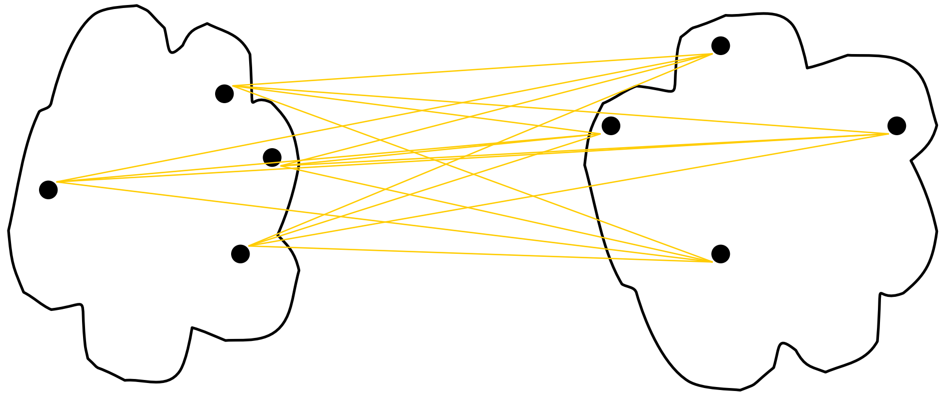
$$-1 \leq s(i) \leq 1$$

- For s(i) to be close to 1 we require a(i) << b(i). As a(i) is a measure of how dissimilar i is to its own cluster, a small value means it is well matched.
- Furthermore, a large b(i) implies that i is badly matched to its neighbouring cluster.
- Thus an s(i) close to one means that the data is appropriately clustered.
- If s(i) is close to negative one, then by the same logic we see that i would be more appropriate if it was clustered in its neighbouring cluster.
- An s(i) near zero means that the data is on the border of two natural clusters.

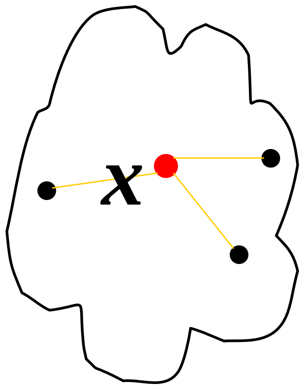
Silhouette Coefficient

- Cohesion: measures how closely related are objects in a cluster
- Separation: measure how distinct or well-separated a cluster is from other clusters


cohesion

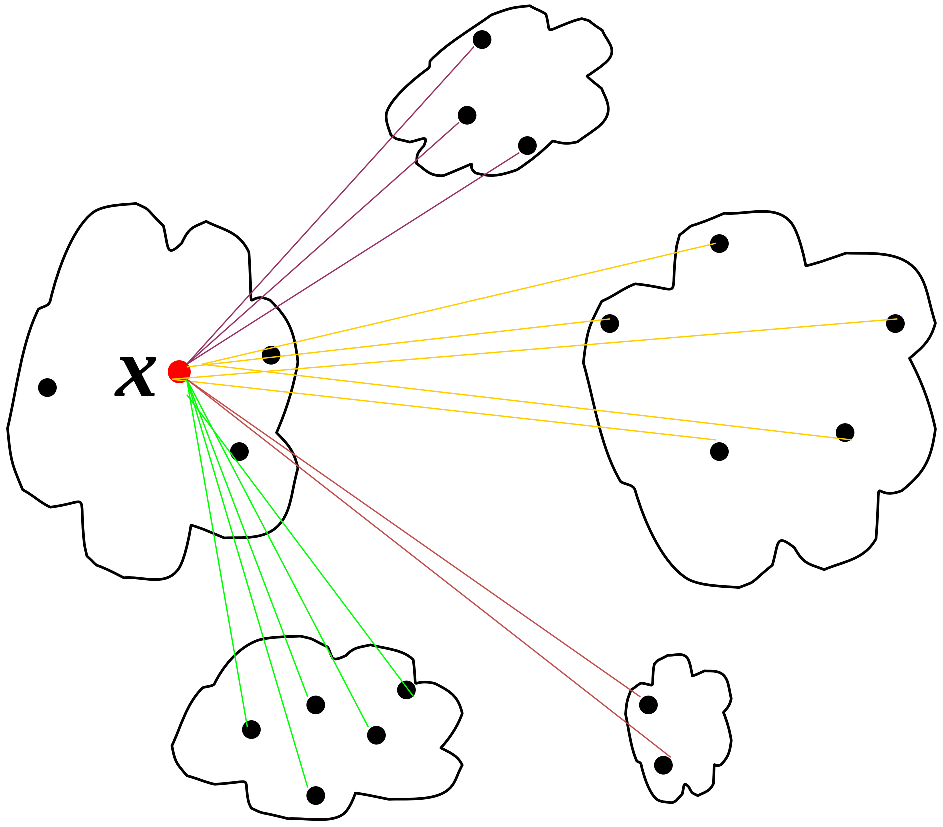

separation

Silhouette Coefficient



cohesion

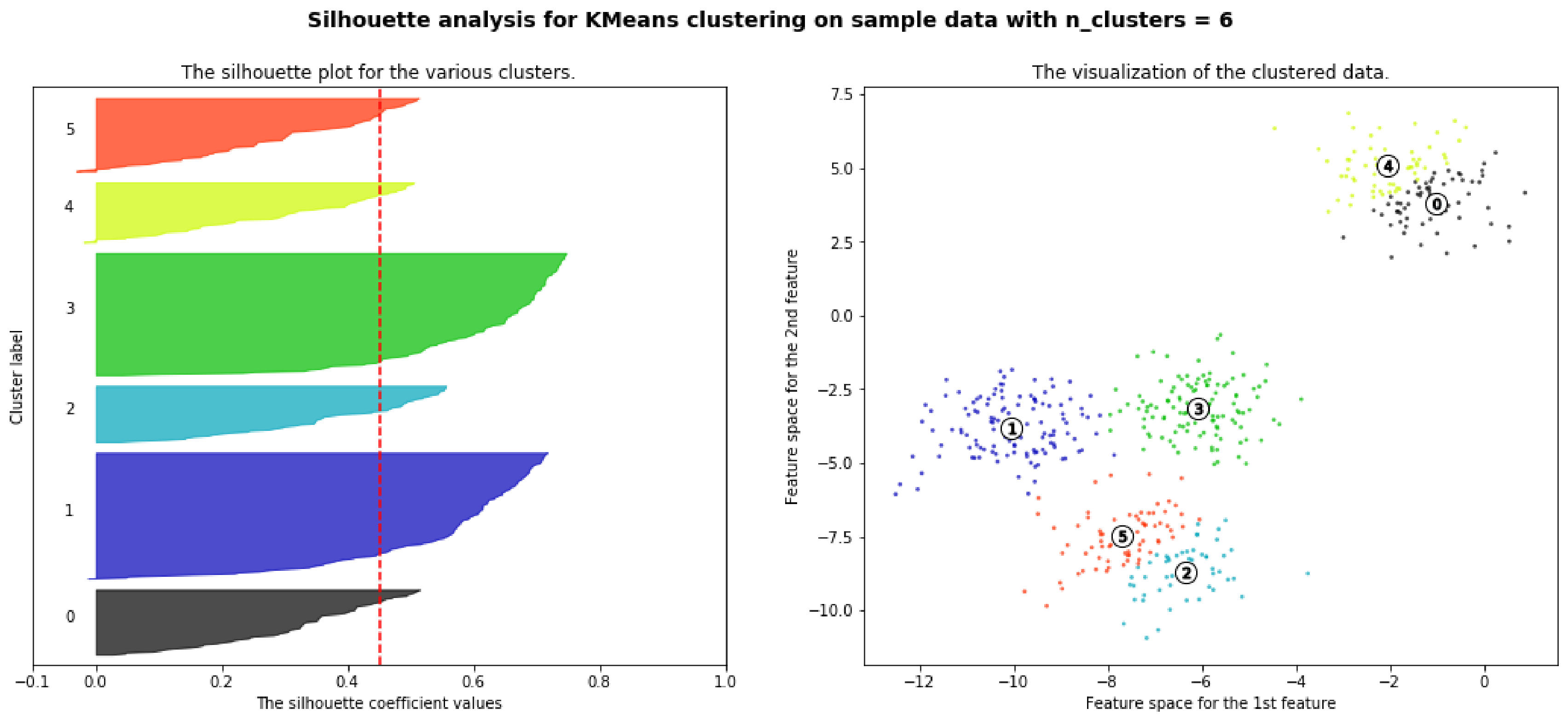
a(*x*): average distance in the cluster



separation

b(x): average distances to others clusters, find minimal

Silhouette Coefficient



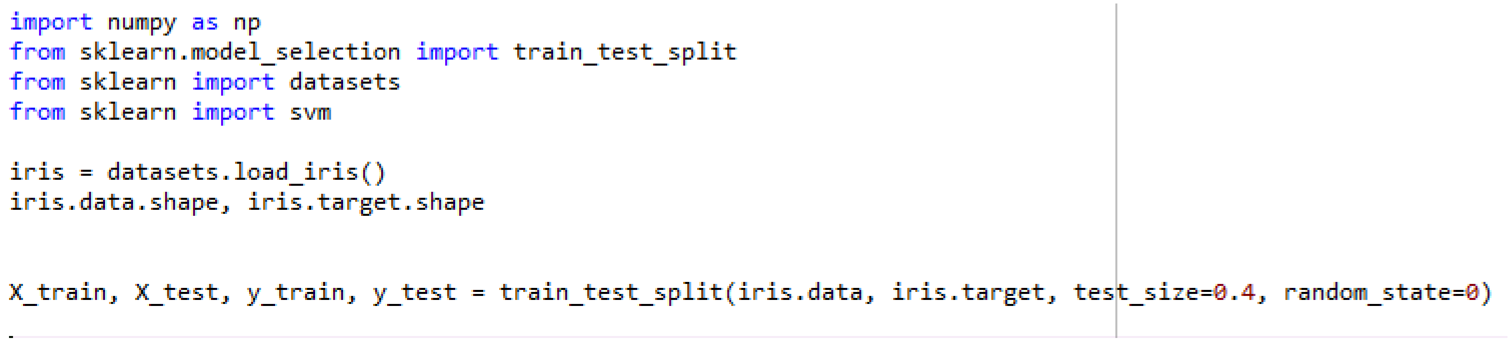
Cross Validation

- Cross-validation, sometimes called rotation estimation, or out-of-sample testing is any of various similar model validation techniques for assessing how the results of a statistical analysis will generalize to an independent data set.
- It is mainly used in settings where the goal is prediction, and one wants to estimate how accurately a predictive model will perform in practice.
- One round of cross-validation involves partitioning a sample of data into complementary subsets, performing the analysis on one subset (called the training set), and validating the analysis on the other subset (called the validation set or testing set).
- Some commonly used cross-validation techniques include k-fold cross validation, where the whole data set is divided into k folds and in each iteration one of these k folds is used as a testing set and all other folds are used a training set.
- This type of validation helps in reducing the bias that is generated due to the uneven distribution in the data.

Cross Validation

In Python there is a library called scikit-learn where we can do a random split into training and test sets quickly computed with the train_test_split helper function.

Let’s load the iris data set where using train_test_split we will divide the data into test and train where my test size will be 40%.



```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn import datasets
from sklearn import svm

iris = datasets.load_iris()
iris.data.shape, iris.target.shape


X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target, test_size=0.4, random_state=0)
```

Cross Validation

When evaluating different settings (“hyperparameters”) for estimators, there is still a risk of overfitting on the test set because the parameters can be tweaked until the estimator performs optimally. This way, knowledge about the test set can “leak” into the model and evaluation metrics no longer report on generalization performance.

To solve this problem, yet another part of the dataset can be held out as “validation set”: training proceeds on the training set, after which evaluation is done on the validation set, and when the experiment seems to be successful, final evaluation can be done on the test set.

However, partitioning the data into three sets will reduce the data drastically for train the model.

A solution to this problem is a procedure called cross-validation (CV for short). A test set should still be held out for final evaluation, but the validation set is no longer needed when doing CV. In the basic approach, called k-fold CV, the training set is split into k smaller sets.

A model is trained using k-1 of the folds as training data;

the resulting model is validated on the remaining part of the data

Cross Validation

The simplest way to use cross-validation is to call the cross_val_score helper function on the estimator and the dataset.

Below example helps us understand how to estimate the accuracy of a linear kernel support vector machine on the iris dataset by splitting the data, fitting a model and computing the score 5 consecutive times:

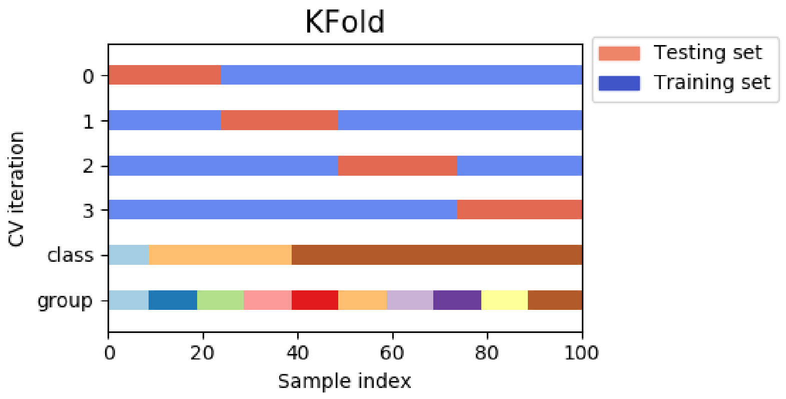
```
from sklearn.model_selection import cross_val_score
clf = svm.SVC(kernel='linear', C=1)
scores = cross_val_score(clf, iris.data, iris.target, cv=5)
scores
```

```
Out[241]: array([0.96666667, 1.        , 0.96666667, 0.96666667, 1.        ])
```

Cross Validation iterators - KFold

KFold divides all the samples in groups of samples, called folds (if , this is equivalent to the Leave One Out strategy), of equal sizes (if possible). The prediction function is learned using folds, and the fold left out is used for test.

```
In [242]: import numpy as np
     ...: from sklearn.model_selection import KFold
     ...:
     ...: X = ["a", "b", "c", "d"]
     ...: kf = KFold(n_splits=2)
     ...: for train, test in kf.split(X):
     ...:     print("%s %s" % (train, test))
[2 3] [0 1]
[0 1] [2 3]
```



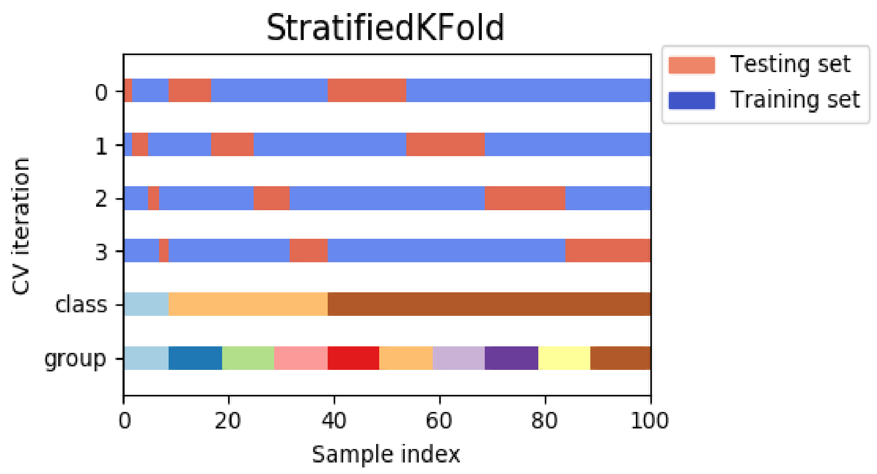
Visualization of the cross-validation behaviour.

KFold is not affected by classes or groups.

Cross Validation iterators – Stratified KFold

StratifiedKFold is a variation of k-fold which returns stratified folds: each set contains approximately the same percentage of samples of each target class as the complete set.

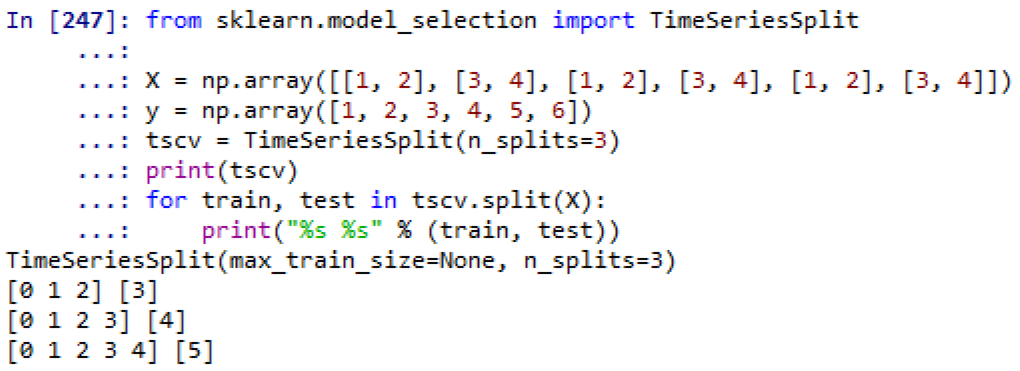
```
In [243]: from sklearn.model_selection import StratifiedKFold
     ...:
     ...: X = np.ones(10)
     ...: y = [0, 0, 0, 0, 1, 1, 1, 1, 1, 1]
     ...: skf = StratifiedKFold(n_splits=3)
     ...: for train, test in skf.split(X, y):
     ...:     print("%s %s" % (train, test))
[2 3 6 7 8 9] [0 1 4 5]
[0 1 3 4 5 8 9] [2 6 7]
[0 1 2 4 5 6 7] [3 8 9]
```



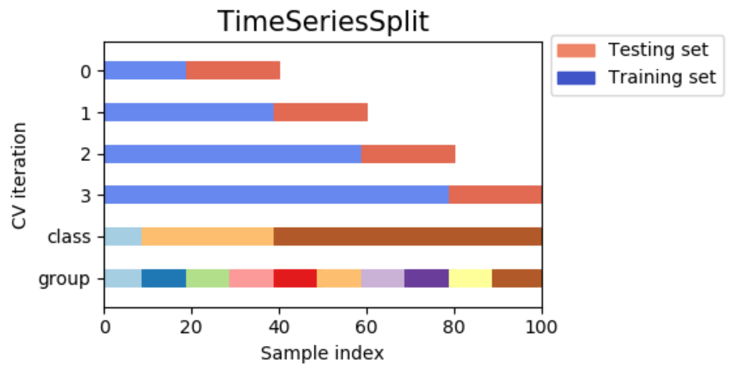
Cross Validation iterators – Time Series split

TimeSeriesSplit is a variation of k-fold which returns first folds as train set and the K+1 th fold as test set.

This class can be used to cross-validate time series data samples that are observed at fixed time intervals.



```
In [247]: from sklearn.model_selection import TimeSeriesSplit
     ...:
     ...: X = np.array([[1, 2], [3, 4], [1, 2], [3, 4], [1, 2], [3, 4]])
     ...: y = np.array([1, 2, 3, 4, 5, 6])
     ...: tscv = TimeSeriesSplit(n_splits=3)
     ...: print(tscv)
     ...: for train, test in tscv.split(X):
     ...:     print("%s %s" % (train, test))
TimeSeriesSplit(max_train_size=None, n_splits=3)
[0 1 2] [3]
[0 1 2 3] [4]
[0 1 2 3 4] [5]
```



Bagging

- Bootstrap aggregating, also called bagging, is a machine learning ensemble meta-algorithm designed to improve the stability and accuracy of machine learning algorithms used in statistical classification and regression. It also reduces variance and helps to avoid over-fitting.
- Bagging was designed to improve classification by combining classifications of randomly generated training sets.
- Given a standard training set D of size n, bagging generates m new training sets D_{i}, each of size n′, by sampling from D uniformly and with replacement. By sampling with replacement, some observations may be repeated in each D_{i}.
- If n′=n, then for large n the set is expected to have the fraction (1 - 1/e) (≈63.2%) of the unique examples of D, the rest being duplicates.
- This kind of sample is known as a bootstrap sample. The m models are fitted using the above m bootstrap samples and combined by averaging the output (for regression) or voting (for classification).

Bagging: Bootstrap aggregating

- Each model in the ensemble votes with equal weight
- Train each model with a random training set
- Random forests do better than bagged entropy reducing DTs

Bootstrap estimation

- Repeatedly draw *n* samples from *D*
- For each set of samples, estimate a statistic
- The bootstrap estimate is the mean of the individual estimates
- Used to estimate a statistic (parameter) and its variance

- For i = 1 .. M
  - Draw $n^* < n$ samples from *D* with replacement
  - Learn classifier C_i
- Final classifier is a vote of $C_1 .. C_M$
- Increases classifier stability/reduces variance

Boosting

- Boosting is a machine learning ensemble meta-algorithm for primarily reducing bias, and also variance in supervised learning, and a family of machine learning algorithms that convert weak learners to strong ones.
- Boosting algorithms consist of iteratively learning weak classifiers with respect to a distribution and adding them to a final strong classifier.
- When they are added, they are typically weighted in some way that is usually related to the weak learners' accuracy. After a weak learner is added, the data weights are readjusted, known as "re-weighting".
- Misclassified input data gain a higher weight and examples that are classified correctly lose weight. Thus, future weak learners focus more on the examples that previous weak learners misclassified.
- AdaBoost is very popular and the most significant historically as it was the first algorithm that could adapt to the weak learners.

Boosting

- Incremental
- Build new models that try to do better on previous model's mis-classifications
    - Can get better accuracy
    - Tends to overfit
- Adaboost is canonical boosting algorithm

- Randomly select $n_1 < n$ samples from D without replacement to obtain D_1
    - Train weak learner C_1
- Select $n_2 < n$ samples from D with half of the samples misclassified by C_1 to obtain D_2
    - Train weak learner C_2
- Select all samples from D that C_1 and C_2 disagree on
    - Train weak learner C_3
- Final classifier is vote of weak learners

Adaboost

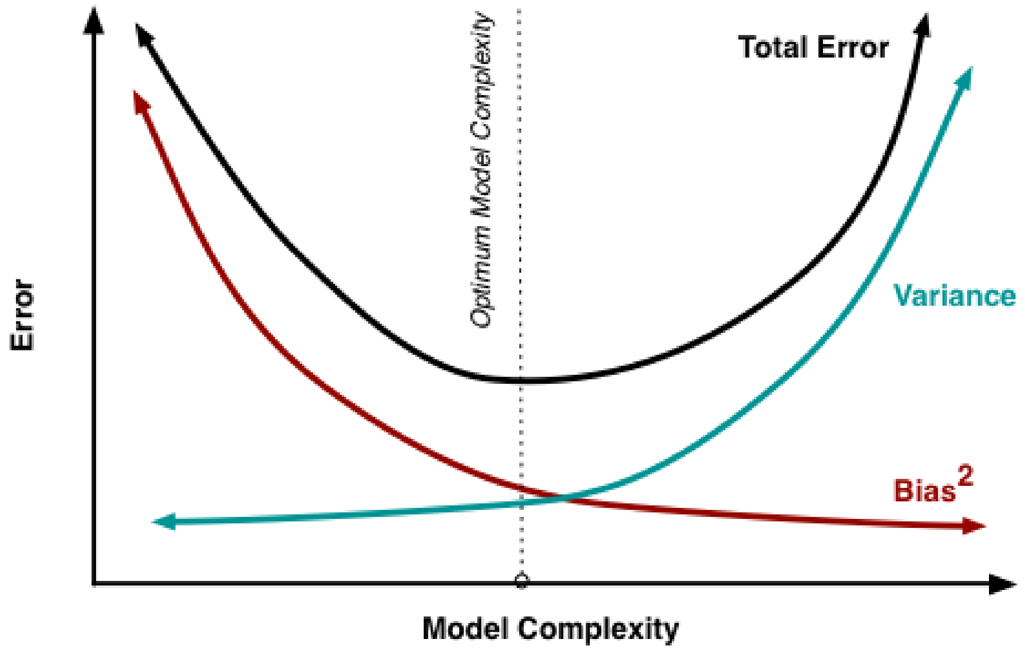
- Learner = Hypothesis = Classifier
- Weak Learner: < 50% error over any distribution
- Strong Classifier: thresholded linear combination of weak learner outputs

Bias v/s Variance

- In machine learning reducible error can be further decomposed into “error due to squared bias” and “error due to variance.”
- The error due to squared bias is the amount by which the expected model prediction differs from the true value or target, over the training data.
- The error due to variance is the amount by which the prediction, over one training set, differs from the expected predicted value, over all the training sets.
- Bias is introduced at the time of model selection, for example if the target truth is highly nonlinear, and you select a linear model to approximate it, then you’re introducing a bias resulting from the linear model’s inability to capture nonlinearity.
- Variance on the other hand, measures how inconsistent are the predictions from one another, over different training sets, not whether they are accurate or not.
- Models that exhibit small variance and high bias under-fit the truth target. Models that exhibit high variance and low bias over-fit the truth target.

Bias v/s Variance

- The “trade-off” between bias and variance can be viewed in this manner – a learning algorithm with low bias must be “flexible” so that it can fit the data well. But if the learning algorithm is too flexible, it will fit each training data set differently, and hence have high variance.



Bias v/s Variance - Regression

For regression, we can easily decompose the error of the learned model into two parts: bias (error 1) and variance (error 2)

Bias: the class of models can’t fit the data.

Fix: a more expressive model class.

Variance: the class of models could fit the data, but doesn’t because it’s hard to fit.

Fix: a less expressive model class.

Bias v/s Variance – decomposition of error

$$E_{D,\varepsilon}\left\{\left(f(x)+\varepsilon-h_D(x)\right)^2\right\}$$

dataset and noise

true function

learned from D

noise

Fix test case *x,* then do this experiment:

1. Draw size *n* sample D=$(x_1,y_1),\ldots.(x_n,y_n)$

2. Train linear regressor h_D *using* D

3. Draw one test example *(x, f(x)+ε)*

4. Measure squared error of h_D on that example *x*

What's the expected error?

Bias v/s Variance – decomposition

- This is something real that you can (approximately) measure experimentally
    - if you have synthetic data
- Different learners and model classes have different *tradeoffs*
    - large bias/small variance: few features, highly regularized, highly pruned decision trees, large-k k-NN…
    - small bias/high variance: many features, less regularization, unpruned trees, small-k k-NN…

Bias v/s Variance - Classification

For classification, we can also decompose the error of a learned classifier into two terms: bias and variance

Bias: the class of models can’t fit the data.

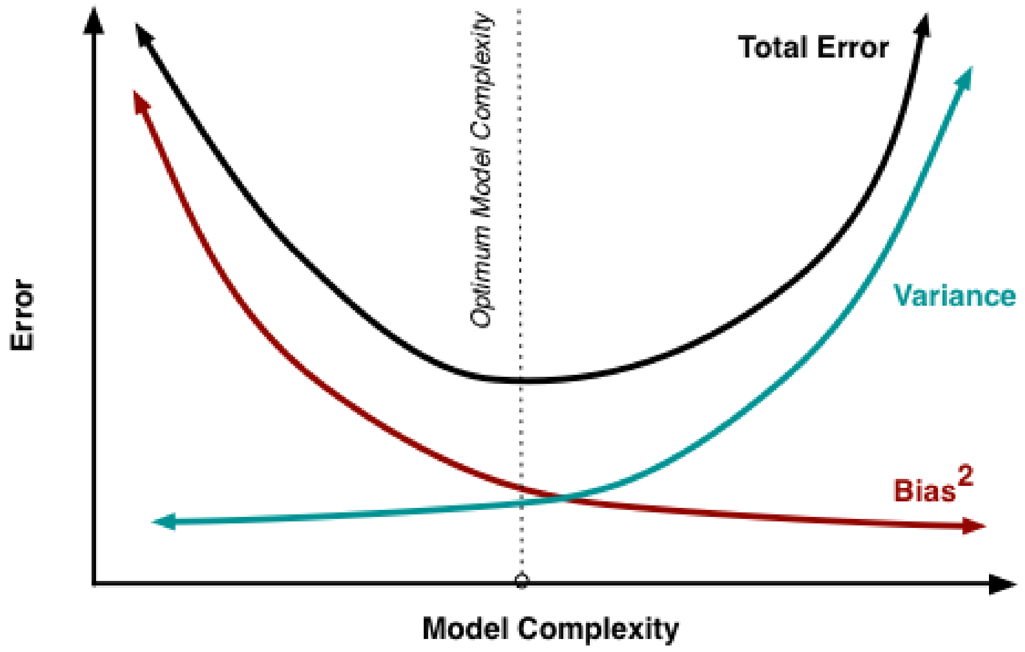
Fix: a more expressive model class.

Variance: the class of models could fit the data, but doesn’t because it’s hard to fit.

Fix: a less expressive model class.

Bias v/s Variance

- The "trade-off" between bias and variance can be viewed in this manner – a learning algorithm with low bias must be "flexible" so that it can fit the data well. But if the learning algorithm is too flexible, it will fit each training data set differently, and hence have high variance.



Cognixia

A Collabera LEARNING SOLUTIONS COMPANY

THANK YOU

www.cognixia.com