SHOGUN-tutorial

Sergey Lisitsyn   Heiko Strathmann   Chiyuan Zhang

March 5, 2013

Contents

Todo list

Part I

Essentials

Chapter 1

Learning

In this part we outline essentials of machine learning.

1.1 Learning is a search process

1.2 Empirical risk minimization (ERM) principle

1.3 Structural risk minimization (SRM) principle

1.4 Linear models

1.5 Supervised learning

1.5.1 Classification

1.5.2 Regression

1.6 Unsupervised learning

1.6.1 Clustering

1.6.2 Dimensionality reduction

1.7 Transfer learning

1.7.1 Multitask learning

1.7.2 Domain adaptation

Part II

Objects in SHOGUN

Chapter 2

Kernels

In this chapter, we describe how kernels are represented in SHOGUN and provide a list of implemented ones.

Chapter 3

Data Representations – Features

In this chapter, we describe how data or features are represented in SHOGUN and provide a list of implemented ones.

Document features framework

3.1 Dense Features

Document dense feat

3.2 Streaming Features

Document streaming tures. H.S.

Part III

Algorithms

Chapter 4

Multiclass learning

In this chapter we describe multiclass learning algorithms available in the SHOGUN toolbox. Multiclass learning refers to the problem with the output space $\mathcal{Y} = \{1, \ldots, K\}$[1], where $K > 2$. Most of real world machine learning classification problems are naturally multiclass. Typical examples include document categorization, image classification, hand-written digit recognition, etc.

Generally, no assumption of any specific structure for the set $\mathcal{Y}$ are made in multiclass learning. When priori knowledge are available for a rich structure of $\mathcal{Y}$, *structured-output learning* algorithms are usually used instead.

Many algorithms, like K-Nearest Neighbors and Naive Bayes, handle both multiclass problems and binary problems naturally (and in an uniform way). Those are described in section 4.1. Section 4.2 describes reduction from multiclass problems into binary problems. Tree-styled classifiers are described in section 4.3.

Several standard datasets are used by examples in this chapter. We summarize them in Table 4.1. All of those datasets can be found in `http://mldata.org`.

4.1 Natural Multiclass Algorithms

4.1.1 K-Nearest Neighbors

K-Nearest Neighbors (KNN) is a very simple and effective algorithm. The learning step actually does nothing but memorizing all the training points and the associated labels. The prediction is carried out by finding the K nearest neighbors of the query point, and then voting. Here K is a hyper-parameter for the algorithm. Smaller K gives the model low bias but high variance; while larger K gives low variance but high bias.

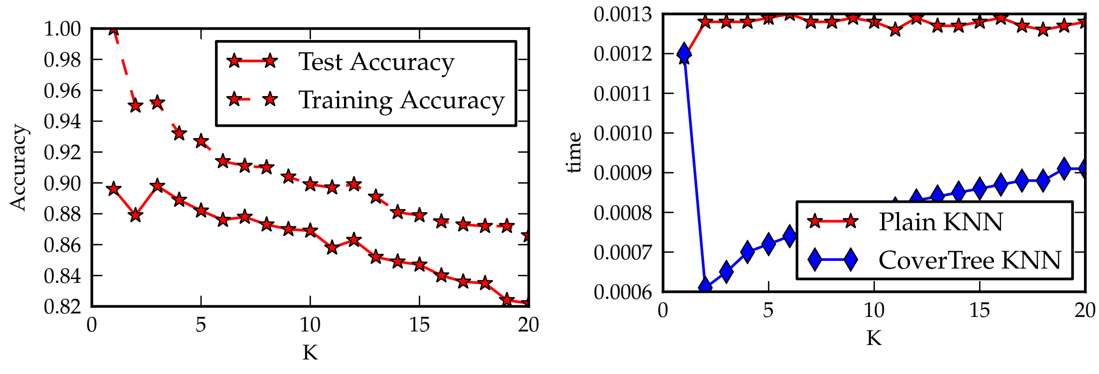
KNN has attracted focus from both industrial and academia ever since its conception. It is easy to implement, and can handle almost arbitrarily complex problem by adjusting one single parameter K. Besides, it also has many nice theoretical properties [Devroye et al., 1996].

In SHOGUN, you can use `CKNN` to perform KNN learning. To construct a KNN machine, you must choose the hyper-parameter K and a distance function. Usually, we simply use the `CEuclideanDistance`, but in general, any subclass of `CDistance` can be used. For demonstration, we select a random subset of 1000 samples from USPS, and run 2-fold cross validation of

[1]Note while we describe the class numbers as from 1 to K, the multiclass machines in SHOGUN expect the examples to be labeled with $0, \ldots, K-1$.

Name	# Classes	# Samples	# Attributes	Remarks
USPS	10	9298	256	Hand-written Digits

Table 4.1: Standard datasets for multiclass learning used in examples.



(a) KNN classification accuracy on USPS.

(b) Prediction time (per example) with and without CoverTree.

Figure 4.1: KNN classification on a random subset (1000 samples) of USPS.

KNN on it with varying K. The accuracy is shown in Fig. 4.1a.

In SHOGUN, you can also use *Cover Tree* [Beygelzimer et al., 2006] to speed up the nearest neighbor searching process in KNN. Just call `set_use_covertree` on the KNN machine to enable or disable this feature. The prediction time comparison for this experiment with and without Cover Tree are shown in Fig. 4.1b.

Although simple and elegant, KNN is generally very resource costly. Because all the training samples are to be memorized literally, the memory cost of KNN "learning" becomes prohibitive when the dataset is huge. Even though the memory is big enough to hold all the data, the prediction will be slow, since the distances between the query point and all the training points need to be computed and ranked. The situation becomes worse if in addition the data samples are all very high-dimensional.

4.1.2 Naive Bayes

Naive Bayes is a simple and fast algorithm for multiclass learning. Formally, it predict the class by computing the posterior probability of each class k after observing the input x:

$$P\left(Y = k | X = x\right) = \frac{P(X = x | Y = k)P(Y = k)}{P(X = x)}$$

The prediction is then made by

$$y = \operatorname*{arg\,max}_{k \in \{1,\ldots,K\}} P(Y = k | X = x)$$

Since $P(X = x)$ is a constant factor for all $P(Y = k | X = x)$, $k = 1, \ldots, K$, there is no need to compute it.

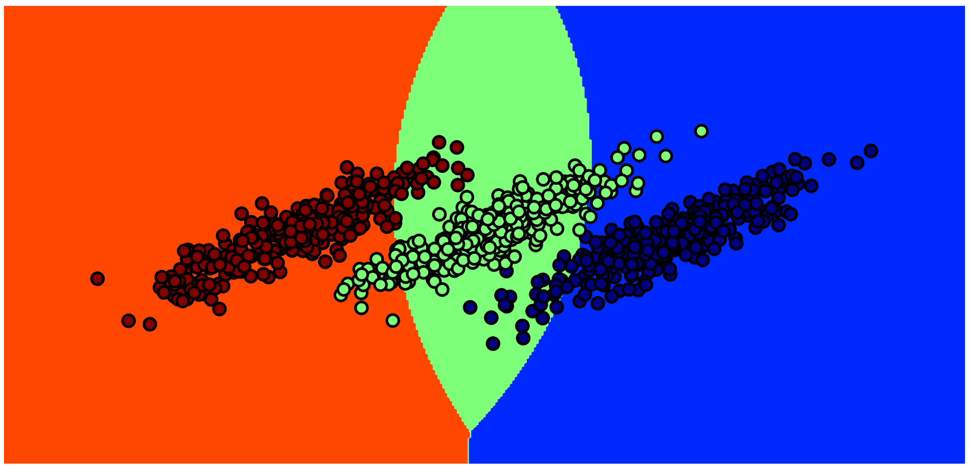


Figure 4.2: Gaussian Naive Bayes fails to learn on a simple 2D example with 3 linearly separable classes.

In SHOGUN, `CGaussianNaiveBayes` implements the Naive Bayes algorithm. It is prefixed with "Gaussian" because the probability model for $P(X = x|Y = k)$ for each k is taken to be a multi-variate Gaussian distribution. Furthermore, each dimension of the feature vector X is assumed to be independent. The "Naive" independence assumption enables us the learn the model by estimating the parameters for each feature dimension independently, thus the whole learning algorithm runs very quickly. And this is also the reason for its name. However, this assumption can be very restrictive. In Fig. 4.2, we show a simple 2D example. There are 3 linearly separable classes. The scattered points are training samples with colors indicating their labels. The filled area indicate the hypothesis learned by the `CGaussianNaiveBayes`. The training samples are actually generated from three Gaussian distributions. But since the covariance for those Gaussian distributions are not diagonal (i.e. there are "rotations"), the GNB algorithm cannot handle them properly.

Although the independent assumption is usually considered to be too optimistic in reality, Naive Bayes sometimes works very well in some applications. For example, in email spam filtering, Naive Bayes[2] is a very popular and widely used method.

This algorithm is closely related to the *Gaussian Mixture Model* (GMM) learning algorithm. However, while GMM is an unsupervised learning algorithm, Gaussian Naive Bayes is supervised learning. It uses the training labels to directly estimate the Gaussian parameters for each class, thus avoids the iterative *Expectation Maximization* procedures in GMM.

The merit of GNB is that both training and predicting are very fast, and it has no hyper-parameters.

[2]More specifically, the discrete Naive Bayes is generally used in this scenario. The main difference with Gaussian Naive Bayes is that a tabular instead of a parametric Gaussian distribution is used to describe the likelihood $P(X = x|K = k)$.

4.1.3 Logistic Regression

Although named logistic *regression*, it is actually a classification algorithm. Similar to *Naive Bayes*, logistic regression computes the posterior $P(Y = k|X = x)$ and makes prediction by

$$y = \operatorname*{arg\,max}_{k\in\{1,\dots,K\}} P(Y = k|X = x)$$

However, Naive Bayes is a *generative model*, in which the distribution of the input variable X is also modeled (by a Gaussian distribution in this case). But logistic regression is a *discriminative model*, which doesn't care about the distribution of X, and models the posterior directly. Actually, the two algorithms are a *generative-discriminative pair* [Ng et al., 2001].

To be specific, logistic regression uses *linear* functions in X to model the posterior probabilities:

$$\log \frac{P(Y = 1|X = x)}{P(Y = K|X = x)} = \beta_{10} + \beta_1^T x \tag{4.1}$$

$$\log \frac{P(Y = 2|X = x)}{P(Y = K|X = x)} = \beta_{20} + \beta_2^T x \tag{4.2}$$

$$\vdots$$

$$\log \frac{P(Y = K-1|X = x)}{P(Y = K|X = x)} = \beta_{(K-1)0} + \beta_{K-1}^T x \tag{4.3}$$

The training of a logistic regression model is carried out via *maximum likelihood estimation* of the parameters $\boldsymbol{\beta} = \{\beta_{10}, \beta_1^T, \dots, \beta_{(K-1)0}, \beta_{K-1}^T\}$. There is no closed form solution for the estimated parameters.

There is not independent implementation of logistic regression in SHOGUN, but the `CLibLinear` becomes a logistic regression model when constructed with the argument `L2R_LR`. This model also include a regularization term of the ℓ_2-norm of $\boldsymbol{\beta}$. If sparsity in $\boldsymbol{\beta}$ is needed, one can also use `L1R_LR`, which replaces the ℓ_2-norm regularizer with a ℓ_1-norm regularizer.

Unfortunately, the logistic regression in SHOGUN does not support multiclass problem yet.

4.2 Reduction to Binary Problems

Since binary classification problems are one of the most thoroughly studied problems in machine learning, it is very appealing to consider reducing multiclass problems to binary ones. Then many advanced learning and optimization techniques as well as generalization bound analysis for binary classification can be utilized.

In SHOGUN, the strategies of reducing a multiclass problem to binary classification problems are described by an instance of `CMulticlassStrategy`. A multiclass strategy describes

1. How to train the multiclass machine as a number of binary machines?
   - How many binary machines are needed?
   - For each binary machine, what subset of the training samples are used, and how are they colored[3]?

[3]In multiclass problems, we use *coloring* to refer partitioning the classes into two groups: $+1$ and -1, or black and white, or any other meaningful names.

Strategy	Training Time	Test Time	Accuracy
One-vs-Rest	1.72	2.25	92.05%
One-vs-One	2.14	4.45	93.75%

Table 4.2: Comparison of One-vs-Rest and One-vs-One multiclass reduction strategy on the USPS dataset.

2. How to combine the prediction results of binary machines into the final multiclass prediction?

The user can derive from the virtual class `CMulticlassStrategy` to implement a customized multiclass strategy. But usually the built-in strategies are enough for general problems. We will describe the built-in *One-vs-Rest*, *One-vs-One* and *Error-Correcting Output Codes* strategies in the following subsections.

The basic routine to use a multiclass machine with reduction to binary problems in shogun is to create a generic multiclass machine and then assign a particular multiclass strategy and a base binary machine.

4.2.1 One-vs-Rest and One-vs-One

The *One-vs-Rest* strategy is implemented in `CMulticlassOneVsRestStrategy`. As indicated by the name, this strategy reduce a K-class problem to K binary sub-problems. For the k-th problem, where $k \in \{1, \ldots, K\}$, the samples from class k are colored as $+1$, and the samples from other classes are colored as -1. The multiclass prediction is given as

$$f(x) = \arg\max_{k \in \{1,\ldots,K\}} f_k(x)$$

where $f_k(x)$ is the prediction of the k-th binary machines.

The One-vs-Rest strategy is easy to implement yet produces the good performance in many cases. One interesting paper [Rifkin and Klautau, 2004] shows that the One-vs-Rest strategy can be

> *as accurate as any other approach, assuming that the underlying binary classifiers are well-tuned regularized classifiers such as support vector machines.*

Implemented in `CMulticlassOneVsOneStrategy`, the *One-vs-One* strategy [Hastie and Tibshirani, 1997] is another simple and intuitive strategy: it basically produces one binary problem for each pair of classes. So there will be $\binom{K}{2}$ binary problems. At prediction time, the output of every binary classifiers are collected to do voting for the K classes. The class with the highest vote becomes the final prediction.

Compared with the One-vs-Rest strategy, the One-vs-One strategy is usually more costly to train and evaluate because more binary machines are used.

In the following, we demonstrate how to use SHOGUN's One-vs-Rest and One-vs-One multiclass learning strategy on the USPS dataset. For demonstration, we randomly 200 samples from each class for training and 200 samples from each class for testing.

How to organize and reference example code for tutorial?

The CLibLinear is used as the base binary classifier in a CLinearMulticlassMachine, with One-vs-Rest and One-vs-One strategies. The running time and performance is reported in Table 4.2.

4.2.2 Error-Correcting Output Codes

Error-Correcting Output Codes (ECOC) [Dietterich and Bakiri, 1995; Allwein et al., 2000] is a generalization of the One-vs-Rest and One-vs-One strategies. For example, we can represent the One-vs-Rest strategy with the following $K \times K$ coding matrix, or a codebook:

$$\begin{bmatrix} +1 & -1 & -1 & \dots & -1 & -1 \\ -1 & +1 & -1 & \dots & -1 & -1 \\ -1 & -1 & +1 & \dots & -1 & -1 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ -1 & -1 & -1 & \dots & +1 & -1 \\ -1 & -1 & -1 & \dots & -1 & +1 \end{bmatrix}$$

Denote the codebook by B, there is one column of the codebook associated with each of the K classes. For example, the code for class 1 is $[+1, -1, -1, \dots, -1]$. Each row of the codebook corresponds to a binary coloring of all the K classes. For example, in the first row, the class 1 is colored as $+1$, while the rest of the classes are all colored as -1. Associated with each row, there is a binary classifier trained according to the coloring. For example, the binary classifier associated with the first row is trained by treating all the examples of class 1 as positive examples, and all the examples of the rest of the classes as negative examples.

In this special case, there are K rows in the codebook. The number of rows in the codebook is usually called the *code length*. As we can see, this codebook exactly describes how the One-vs-Rest strategy trains the binary sub-machines.

A further generalization is to allow 0-values in the codebook. A 0 for a class k in a row means we ignore (the examples of) class k when training the binary classifiers associated with this row. With this generalization, we can also easily describes the One-vs-One strategy with a $\binom{K}{2} \times K$ codebook:

$$\begin{bmatrix} +1 & -1 & 0 & \dots & 0 & 0 \\ +1 & 0 & -1 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & 0 \\ +1 & 0 & 0 & \dots & -1 & 0 \\ 0 & +1 & -1 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & 0 & \dots & +1 & -1 \end{bmatrix}$$

Here each of the $\binom{K}{2}$ rows describes a binary classifier trained with a pair of classes. The resultant binary classifiers will be identical as those described by a One-vs-One strategy.

Since 0 is allowed in the codebook to ignore some classes, this kind of codebooks are usually called *sparse codebook*s, while the codebooks with only $+1$ and -1 are usually called *dense codebook*.

In general case, we can specify any code length and fill the codebook arbitrarily. However, some rules should be followed:

1. Each row must describe a *valid* binary coloring. In other words, both $+1$ and -1 should appear at least once in each row. Or else a binary classifier cannot be obtained for this row.

2. It is good to avoid duplicated rows. There is generally no harm to have duplicated rows, but the resultant binary classifiers are completely identical provided the training algorithm for the binary classifiers are deterministic. So this can be a waste of computational resource.

3. Negative rows are also duplicated. Simply inversing the sign of a code row does not produce a "new" code row. Because the resultant binary classifier will simply be the negative classifier associated with the original row.

Though you can certainly generate your own codebook, it is usually easier to use the SHOGUN built-in procedures to generate codebook automatically. There are various codebook generators (called *encoders*) in SHOGUN. However, before describing those encoders in details, let us notice that a codebook only describes how the sub-machines are trained. But we still need a way to specify how the binary classification results of the sub-machines can be combined to get a multiclass classification result.

Review the codebook again: corresponding to each class, there is a column. We call the codebook column the (binary) *code* for that class. For a new sample x, by applying the binary classifiers associated with each row successively, we get a prediction vector of the same length as the *code*s. Deciding the multiclass label from the prediction vector (called *decoding*) can be done by minimizing the *distance* between the codes and the prediction vector. Different *decoders* define different choices of distance functions. For this reason, it is usually good to make the mutual distance between codes of different classes large. In this way, even though several binary classifiers make wrong predictions, the distance of the resultant prediction vector to the code of the *true* class is likely to be still smaller than the distance to other classes. So correct results can still be obtained even when some of the binary classifiers make mistakes. This is the reason for the name *Error-Correcting Output Codes*.

In SHOGUN, encoding schemes are described by subclasses of `CECOCEncoder`, while decoding schemes are described by subclasses of `CECOCDecoder`. Theoretically, any combinations of encoder-decoder pairs can be used. Here we will introduce several common encoder/decoders in shogun.

- `CECOCRandomDenseEncoder`: This encoder generate random dense ($+1/-1$) codebooks and choose the one with the largest *minimum mutual distance* among the classes. The recommended code length for this encoder is $10 \log K$ [Allwein et al., 2000].

- `CECOCRandomSparseEncoder`: This is similar to the random dense encoder, except that sparse ($+1/-1/0$) codebooks are generated. The recommended code length for this encoder is $15 \log K$ [Escalera et al., 2009].

- `CECOCOVREncoder`, `CECOCOVOEncoder`: These two encoders mimic the One-vs-Rest and One-vs-One strategies respectively. They are implemented mainly for demonstrative purpose. When suitable decoders are used, the results will be equivalent to the corresponding strategies, respectively.

- `CECOCDiscriminantEncoder`

- `CECOCForestEncoder`

Describe the name ECOC here, and describe the SHOGUN ECOC encoding/decoding pairs.

4.3 Tree-style Algorithms

Chapter 5

Statistical Testing

This chapter describes SHOGUN's framework for statistical hypothesis testing. We begin by giving a brief outline of the problem setting in section 5.1. Then, we describe methods for two-sample testing for independence testing.

Methods for two-sample testing currently consist of tests based on the *Maximum Mean Discrepancy*, section 5.2. There are two types of tests available, a quadratic time test, which is described in section 5.2.1; and a linear time test, which is described in section 5.2.2. Both come in various flavours.

Independence testing is currently based in the *Hilbert Schmidt Independence Criterion*, which is described in section 5.3 along with a test using it.

5.1 Statistical Hypothesis Testing

To set the context, we here briefly describe statistical hypothesis testing. Informally, one defines a hypothesis on a certain domain and then uses a statistical test to check whether this hypothesis is true. Formally, the goal is to reject a so-called *null-hypothesis* H_0, which is the complement of an *alternative-hypothesis* H_A.

To distinguish the hypothesises, a test statistic is computed on sample data. Since sample data is finite, this corresponds to sampling the true distribution of the test statistic. There are two different distributions of the test statistic – one for each hypothesis. The *null-distribution* corresponds to test statistic samples under the model that H_0 holds; the *alternative-distribution* corresponds to test statistic samples under the model that H_A holds.

In practice, one tries to compute the quantile of the test statistic in the null-distribution. In case the test statistic is in a high quantile, i.e. it is unlikely that the null-distribution has generated the test statistic – the null-hypothesis H_0 is rejected.

There are two different kinds of errors in hypothesis testing:

- A *type I error* is made when $H_0 : p = q$ is wrongly rejected. That is, the tests says that the samples are from different distributions when they are not.

- A *type II error* is made when $H_A : p = q$ is wrongly accepted. That is, the tests says that the samples are from same distributions when they are from the same.

A so called *consistent* test achieves zero type II error for a fixed type I error.

To decide whether to reject H_0, one could set a threshold, say at the 95% quantile of the null-distribution, and reject H_0 when the test statistic lies below that threshold. This means that the chance that the samples were generated under H_0 are 5%. We call this number the test power α (in this case $\alpha = 0.05$). It is an upper bound on the probability for a type I error. An alternative way is simply to compute the quantile of the test statistic in the null-distribution, the so-called *p-value*, and to compare the p-value against a desired test power, say $\alpha = 0.05$, by hand. The advantage of the second method is that one not only gets a binary answer, but also an upper bound on the type I error.

In order to construct a two-sample test, the null-distribution of the test statistic has to be approximated. One way of doing this for any two-sample test is called *bootstrapping*:

Algorithm 5.1 Bootstrapping a null-distribution.

Inputs are:

- X, Y, sets of samples from p, q of size m, n respectively

Output is:

- One sample from null-distribution. Simply repeat for more samples.

```
1: $Z \leftarrow \{X, Y\}$
2: $\hat{Z} = \{\hat{z}_1, ..., \hat{z}_{m+n}\} \leftarrow \text{randperm}(Z)$        (generate a random ordering)
3: $\hat{X} \leftarrow \{\hat{z}_1, ...\hat{z}_m\}$
4: $\hat{Y} \leftarrow \{\hat{z}_{m+1}, ...\hat{z}_{m+n}\}$
5: return Test statistic for $\hat{X}, \hat{Y}$
```

Bootstrapping is a useful technique to create ground-truth samples from a null-distribution. However, it is rather costly because the statistic has to be re-computed for every sample. More details will be given when individual tests are described.

Interface for Statistical Testing

SHOGUN implements statistical testing in the abstract class `CTestStatistic`.

- Test statistics can be computed with `compute_statistic`.
- P-values for a given statistic can be computed via `compute_p_value`. Results depend on method that is set for approximating null-distribution.
- Statistic thresholds for a given p-value can be computed via `compute_threshold`. Results depend on method that is set for approximating null-distribution.
- A number of samples can be drawn from the null-distribution using bootstrapping via `bootstrap_null`. This will call `compute_statistic` a certain number of times while underlying data is modified in such way that the null hypothesis $H_0 : p = q$ is true.
- A complete two-sample test can be computed using `perform_test`. There are two different versions of this method:
    - One without parameters which computes the statistic and returns a p-value for it. This p-value can be used to (not) reject the null hypothesis.
    - One which has a test level α as a parameter and returns `true` if the null hypothesis is rejected and `false` otherwise. Obviously, this method is just a simple wrapper of the above one.

The `perform_test` methods are convenience wrappers for `compute_statistic` and `compute_p_value`. However, in certain cases, it might be possible to compute statistic and p-value in the same loop which is more efficient. In subclasses of `CTestStatistic`, the first one might be overwritten in order to implement this – if possible. See class documentations for availability. If an efficient implementation is not existent, `perform_test` simply is a wrapper for `compute_statistic` and `compute_p_value`.

5.2 Two-Sample-Testing with the Maximum Mean Discrepancy

An important class of hypothesis tests are the *two-sample tests*, which will be defined in the following. In two-sample testing, one tries to find out whether to sets of samples come from different distributions. Given two probability distributions p, q and i.i.d. samples $X = \{x_i\}_{i=1}^m \subseteq \mathbb{R}^d \sim p$ and $Y = \{y_i\}_{i=1}^n \subseteq \mathbb{R}^d \sim p$, the two sample test distinguishes the hypothesises

$$H_0 : p = q$$
$$H_A : p \neq q$$

In order to solve this problem, it is desirable to have a criterion than takes a positive unique value if $p \neq q$, and zero if and only if $p = q$. The so called *Maximum Mean Discrepancy* (MMD), has this property and allows to distinguish any two probability distributions, if used in a *reproducing kernel Hilbert space* (RKHS). It is the distance of the mean embeddings μ_p, μ_q of the distributions p, q in such a RKHS $\mathcal{F}$ – which can also be expressed in terms of expectation of kernel functions, i.e.

$$\begin{aligned}\text{MMD}[\mathcal{F}, p, q] &= ||\mu_p - \mu_q||_{\mathcal{F}}^2 \\ &= \mathbf{E}_{x,x'}\left[k(x,x')\right] - 2\mathbf{E}_{x,y}\left[k(x,y)\right] + \mathbf{E}_{y,y'}\left[k(y,y')\right]\end{aligned} \tag{5.1}$$

See [Gretton et al., 2012a, Section 2] for details. We here only describe how to use the MMD for two-sample testing. SHOGUN offers two types of test statistic based on the MMD, one with quadratic costs both in time and space, and on with linear time and constant space costs. Both come in different versions and with different methods how to approximate the null-distribution in order to construct a two-sample test.

5.2.1 Quadratic Time MMD Statistic

We now describe the quadratic time MMD, as described in [Gretton et al., 2012a, Lemma 6], which is implemented in SHOGUN. All methods in this section are implemented in `CQuadraticTimeMMD`, which accepts any type of SHOGUN features.

An unbiased estimate for expression 5.1 can be obtained by estimating expected values with sample means

$$\begin{aligned}\text{MMD}_u^2[\mathcal{F}, X, Y] =& \frac{1}{m(m-1)}\sum_{i=1}^m\sum_{j\neq i}^m k(x_i, x_j) + \frac{1}{n(n-1)}\sum_{i=1}^n\sum_{j\neq i}^n k(y_i, y_j) \\ &- \frac{2}{mn}\sum_{i=1}^m\sum_{j\neq i}^n k(x_i, y_j)\end{aligned}$$

A biased estimate would be

$$\begin{aligned}\mathrm{MMD}_b^2[\mathcal{F},X,Y] =& \frac{1}{m^2}\sum_{i=1}^{m}\sum_{j=1}^{m}k(x_i,x_j)+\frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}k(y_i,y_j)\\ &-\frac{2}{mn}\sum_{i=1}^{m}\sum_{j\neq i}^{n}k(x_i,y_j)\end{aligned}$$

To compute statistic, use `compute_statistic`. To decide which statistic to use, use `set_statistic_type` with arguments `BIASED` or `UNBIASED` to activate this statistic type. Note that some methods for approximating the null-distribution only work with one of both types. Both statistics' computational costs are quadratic both in time and space. Note that the method returns $m\,\mathrm{MMD}_b^2[\mathcal{F},X,Y]$ since null distribution approximations work on m times null distribution.

Bootstrapping

As for any two-sample test in SHOGUN, bootstrapping can be used to approximate the null-distribution with both types of quadratic MMD statistic. This results in a consistent, but slow test. Note that for each sample, the quadratic time estimate has to be recomputed. The number of samples to take is the only parameter. If unsure what to do, bootstrapping is the recommended way of constructing a test. As a rule of thumb, use at least 250 samples. See `bootstrap_null` in `CTwoDistributionsTestStatistic` and `CKernelTwoSampleTestStatistic`. Strongly consider using pre-computed kernel matrices as described in section 5.2.3.

Spectrum Approximation

Approximates the null-distribution using the Eigen-Spectrum of the kernel matrix of the joint samples. Was described in [Gretton et al., 2012b]. This is a fast and consistent test. Effectively, the null-distribution of the biased statistic is sampled, but in a more efficient way than the bootstrapping approach. The converges as

$$m\,\mathrm{MMD}_b^2 \rightarrow \sum_{l=1}^{\infty}\lambda_l z_l^2 \tag{5.2}$$

where $z_l \sim \mathcal{N}(0,2)$ are i.i.d. normal samples and λ_l Eigenvalues of expression 2 in [Gretton et al., 2012b], which can be empirically estimated by $\hat{\lambda}_l = \frac{1}{m}\nu_l$ where ν_l are the Eigenvalues of the centred kernel matrix of the joint samples X and Y. The distribution in expression 5.2 can be easily sampled. SHOGUN's implementation has two parameters:

- Number of samples from null-distribution. The more, the more accurate. As a rule of thumb, use 250.
- Number of Eigenvalues of the Eigen-decomposition of the kernel matrix to use. The more, the better the results get; however, the Eigen-spectrum of the joint gram matrix usually decreases very fast. See [Gretton et al., 2012b] for details.

If the kernel matrices are diagonal dominant, this method is likely to fail. For that and more details, see the original paper. Computational costs are much lower than bootstrapping, which is the only consistent alternative. Since Eigenvalues of the gram matrix has to be computed, costs are in $\mathcal{O}(m^3)$.

To get a number of samples, use `sample_null_spectrum`; to use that method for testing, use `set_null_approximation_method(MMD2_SPECTRUM)`. Both methods are to be found in `CQuadraticTimeMMD`. **Important:** This method only works with the biased statistic.

Gamma Approximation

Another method for approximating the null-distribution is by matching the first two moments of a gamma-distribution and then use that. This is not consistent, but usually also gives good results while being very fast. However, there are distributions where the method fails; therefore, the type I error should always be monitored. Described in [Gretton et al., 2012b]. It uses

Gamma Approximati
Merge with compute
statistic in code and d
ument! H.S.

$$m\,\mathrm{MMD}_b(Z) \sim \frac{x^{\alpha-1}\exp(-\frac{x}{\beta})}{\beta^\alpha\Gamma(\alpha)} \tag{5.3}$$

where

$$\alpha = \frac{(\mathbf{E}(\mathrm{MMD}_b(Z)))^2}{\mathrm{var}(\mathrm{MMD}_b(Z))} \qquad \text{and} \qquad \beta = \frac{m\,\mathrm{var}(\mathrm{MMD}_b(Z))}{(\mathbf{E}(\mathrm{MMD}_b(Z)))^2}$$

Then, any threshold and p-value can be computed using the gamma distribution in expression 5.3. Computational costs are in $\mathcal{O}(m^2)$.

To use that method for testing, use `set_null_approximation_method(MMD2_GAMMA)`, to be found in `CQuadraticTimeMMD`. **Important:** This method only works with the biased statistic.

5.2.2 Linear Time MMD Statistic

We now describe the linear time MMD, as described in [Gretton et al., 2012a, Section 6], which is implemented in SHOGUN. All methods in this section are implemented in `CLinearTimeMMD`.

A fast, unbiased estimate for expression 5.1 which still uses all available data can be obtained by dividing data into two parts and then compute

$$\mathrm{MMD}_l^2[\mathcal{F}, X, Y] = \frac{1}{m_2}\sum_{i=1}^{m_2} k(x_{2i}, x_{2i+1}) + k(y_{2i}, y_{2i+1}) - k(x_{2i}, y_{2i+1}) - k(x_{2i+1}, y_{2i})$$

where $m_2 = \lfloor \frac{m}{2} \rfloor$. While the above expression assumes that m data are available from each distribution, the statistic in general works in an online setting where features are obtained one by one. Since only pairs of four points are considered at once, this allows to compute it on data streams. In addition, the computational costs are linear in the number of samples that are considered from each distribution. These two properties make the linear time MMD very applicable for large scale two-sample tests. In theory, any number of samples can be processed – time is the only limiting factor.

How to Pass Data to `CLinearTimeMMD`

To account for this streaming nature of the linear time MMD, the implementation in SHOGUN is based on the streaming framework around the class `CStreamingFeatures`. The latter basically implements an interface to get data one by one or in larger blocks. See section 3.2 for more details on the streaming interface. If CLinearTimeMMD should be used on non-streaming data, such as for example CDenseFeatures as described in section 3.1, simply construct an instance of CStreamingFeatures from these. For example, CStreamingDenseFeatures offers a constructor to construct an instance from an object of type CDenseFeatures which then can be passed to CLinearTimeMMD. If a streaming feature class for your desired data does not exist or does not offer an constructor to build from an object in memory, write to the mailing list. **Important:** The underlying parser of CDenseFeatures has to be started (or ensured that it is not needed) before CLinearTimeMMD computes anything. Otherwise, a deadlock might occur.

After streaming data is passed to CLinearTimeMMD, all interfaces work as before – with the difference that new data is taken from the stream in every call of `compute_statistic` and related methods. Note that taking data from the stream one by one results in a very inefficient way of computing the statistic due to involved overhead in the framework. To avoid this, there is an optional constructor parameter to specify a blocksize (note there is a default value). This number specifies how many samples are taken from the stream at once. The statistic then is computed in bursts on these blocks. In principle, the blocksize should be as large as possible, however, the memory consumption increases linear in the number of samples per block. If set larger than m, all samples will be processed at once.

Bootstrapping

As for any two-sample test in SHOGUN, bootstrapping can be used to approximate the null distribution. This results in a consistent, but slow test. The number of samples to take is the only parameter. As a rule of thumb, use at least 250 samples. See `bootstrap_null` in CLinearTimeMMD. Note that since CLinearTimeMMD operates on streaming features, *new* data is taken from the stream in every iteration. This is in contrast to the usual way of bootstrapping as described in algorithm 5.1, where data is permuted in every iteration.

Also note that in general, bootstrapping is not really necessary since with the Gaussian approximation, a fast and consistent estimate of the null-distribution is available for the linear time MMD. However, to ensure that the Gaussian approximation is accurate, it should always be checked against bootstrapping at least once.

Gaussian Approximation

Since both the null- and the alternative distribution are Gaussian with equal variance (and different mean), it is possible to approximate the null-distribution by using a linear time estimate for this variance. An unbiased, linear time estimator for

$$\text{var}[\text{MMD}_l^2[\mathcal{F}, X, Y]]$$

can simply be computed by computing the empirical variance of

$$k(x_{2i}, x_{2i+1}) + k(y_{2i}, y_{2i+1}) - k(x_{2i}, y_{2i+1}) - k(x_{2i+1}, y_{2i}) \qquad (1 \leq i \leq m_2)$$

A normal distribution with this variance and zero mean can then be used as an approximation for the null-distribution. This results in a consistent test and is very fast. However, note that it is an approximation and its accuracy depends on the underlying data distributions. It is a good idea to compare to the bootstrapping approach first to determine an appropriate number of samples to use. This number is usually in the tens of thousands.

To use the method for testing, use `set_null_approximation_method(MMD1_GAUSSIAN)`, to be found in CLinearTimeMMD. Using the Gaussian approximation, the null distribution can be estimated on the fly while computing the test statistic. The method compute_statistic_and_variance does exactly this (results written into parameter references). The perform_test methods of CTestStatistic for performing a complete two-sample are overwritten to make use of this more efficient approach. Use these instead of computing statistic and p-value separately.

5.2.3 Precomputed Kernel Matrices for Quadratic Time MMD

For all MMD-based two-sample-tests, elements of kernel matrices of sample data have to be used. By default, all computations are done *in-place* when possible, which means that the underlying kernel is evaluated on the fly (There are exceptions, when the matrix has to be stored, for example in order so solve Eigenvalue problems). However, for the quadratic time MMD, this may be inefficient when statistics are computed multiple times – as in bootstrapping. Therefore, it is possible to initialize CQuadraticTimeMMD with a pre-computed CCusotmKernel. This kernel may be computed from any other kernel by simply passing the latter to the constructor of CCusotmKernel. This should be done whenever the kernel matrix fits into memory; it greatly improves performance. In bootstrapping, the kernel matrix only has to be permuted instead of being re-computed in every iteration. But there is also a (small) advantage for all other methods since SHOGUN computes kernel matrices in multiple threads.

In contrast, CLinearTimeMMD should not be used with CCusotmKernels since it does not even need all elements – so pointless computations would be made. Also, CLinearTimeMMD might be c

5.2.4 Kernel Selection for MMD

SHOGUN's kernel selection methods for MMD based two-sample tests are all based around Sriperumbudur et al. [2009]; Gretton et al. [2012c]. For the CLinearTimeMMD, Gretton et al. [2012c] describes a way of selecting the *optimal* kernel in the sense that the test's type II error is minimised. For the linear time MMD, this is the method of choice. It is done via maximising the MMD statistic divided by its standard deviation and it is possible for single kernels and also for convex combinations of them. For the CQuadraticTimeMMD, the best method in literature is choosing the kernel that maximised the MMD statistic. For convex combinations of kernels, this can be achieved via a $L2$ norm constraint. A detailed comparison of all methods on numerous datasets can be found in Strathmann [2012].

MMD Kernel selection on SHOGUN always involves an implementation of the base class CMMDKernelSelection, which defines the interface for kernel selection. If combinations of kernel should be considered, there is a sub-class CMMDKernelSelectionComb. In addition, it involves setting up a number of baseline kernels $\mathcal{K}$ to choose from/combine in the form of a CCombinedKernel. All methods compute their results for a fixed set of these baseline kernels. We later give an example how to use these classes after providing a list of available methods.

CMMDKernelSelectionMedian: Selects from a set CGaussianKernel instances the one whose width parameter is closest to the median of the pairwise distances in the data. The median is computed on a certain number of points from each distribution that can be specified as a parameter. Since the median is a stable statistic, one does not have to compute all pairwise distances but rather just a few thousands. This method a useful (and fast) heuristic that in many cases gives a good hint on where to start looking for Gaussian kernel widths. It is for example described in [Gretton et al., 2012a]. Note that it may fail badly in selecting a good kernel for certain problems.

`CMMDKernelSelectionMax`: Selects from a set of arbitrary baseline kernels a single one that maximises the used MMD statistic – more specific its estimate.

$$k^* = \arg\max_{k \in \mathcal{K}} \hat{\eta}_k,$$

where η_k is an empirical MMD estimate for using a kernel k. This was first described in Sriperumbudur et al. [2009] and was empirically shown to perform better than the median heuristic above. However, it remains a heuristic that comes with no guarantees. Since MMD estimates can be computed in linear and quadratic time, this method works for both methods. However, for the linear time statistic, there exists a better method.

`CMMDKernelSelectionOpt`: Selects the optimal single kernel from a set of baseline kernels. This is done via maximising the ratio of the linear MMD statistic and its standard deviation.

$$k^* = \arg\max_{k \in \mathcal{K}} \frac{\hat{\eta}_k}{\hat{\sigma}_k + \lambda},$$

where η_k is a linear time MMD estimate for using a kernel k and $\hat{\sigma}_k$ is a linear time variance estimate of η_k to which a small number λ is added to prevent division by zero. These are estimated in a linear time way with the streaming framework that was described in section 5.2.2. Therefore, this method is only available for `CLinearTimeMMD`. Optimal here means that the resulting test's type II error is minimised for a fixed type I error. *Important:* For this method to work, the kernel needs to be selected on *different* data than the test is performed on. Otherwise, the method will produce wrong results.

`CMMDKernelSelectionCombMaxL2`: Selects a convex combination of kernels that maximises the MMD statistic. This is the multiple kernel analogous to `CMMDKernelSelectionMax`. This is done via solving the convex program

$$\boldsymbol{\beta}^* = \min_{\boldsymbol{\beta}} \{\boldsymbol{\beta}^T \boldsymbol{\beta} : \boldsymbol{\beta}^T \boldsymbol{\eta} = \mathbf{1}, \boldsymbol{\beta} \succeq 0\},$$

where $\boldsymbol{\beta}$ is a vector of the resulting kernel weights and $\boldsymbol{\eta}$ is a vector of which each component contains a MMD estimate for a baseline kernel. See [Gretton et al., 2012c] for details. Note that this method is unable to select a single kernel – even when this would be optimal. Again, when using the linear time MMD, there are better methods available.

`CMMDKernelSelectionCombOpt`: Selects a convex combination of kernels that maximises the MMD statistic divided by its covariance. This corresponds to *optimal* kernel selection in the same sense as in class `CMMDKernelSelectionOpt` and is its multiple kernel analogous. The convex program to solve is

$$\boldsymbol{\beta}^* = \min_{\boldsymbol{\beta}} (\hat{Q} + \lambda I) \{\boldsymbol{\beta}^T \boldsymbol{\beta} : \boldsymbol{\beta}^T \boldsymbol{\eta} = \mathbf{1}, \boldsymbol{\beta} \succeq 0\},$$

where again $\boldsymbol{\beta}$ is a vector of the resulting kernel weights and $\boldsymbol{\eta}$ is a vector of which each component contains a MMD estimate for a baseline kernel. The matrix $\hat{Q}$ is a linear time estimate of the covariance matrix of the vector $\boldsymbol{\eta}$ to whose diagonal a small number λ is added to prevent division by zero. See [Gretton et al., 2012c] for details. In contrast to `CMMDKernelSelectionCombMaxL2`, this method is able to select a single kernel when this gives a lower type II error than a combination. In this sense, it contains `CMMDKernelSelectionOpt`.

Procedure In order to use one of the above methods for kernel selection, one has to create a new instance of `CCombinedKernel` and use the method `append_kernel` to append all desired baseline kernels to it. This combined kernel is then passed to the MMD class. Then, an object of any of the above kernel selection methods is created and the MMD instance is passed to it in the constructor. There are then multiple methods to call

- `compute_measures` returns a vector kernel selection criteria if a single kernel selection method is used. It will return a vector of selected kernel weights if a combined kernel selection method is used. For `CMMDKernelSelectionMedian`, the method does throw an error.
- `select_kernel` returns the selected kernel of the method. For single kernels this will be one of the baseline kernel instances. For the combined kernel case, this will be the underlying `CCombinedKernel` instance where the subkernel weights are set to the weights that were selected by the method.

In order to utilise the selected kernel, it has to be passed to an MMD instance via `set_kernel` or the constructor. See the examples for more details.

Examples

There are graphical python examples which plot example data, alternative and null-distributions. See figures 5.2 and 5.1 for a screenshot for quadratic and linear time MMD respectively. In addition there are examples for linear time and quadratic time MMD that illustrate how to estimate type I and type II errors rates, and how to use kernel selection methods.

5.3 Independence Testing with the HSIC Statistic

Independence testing tries to solve the following problem (taken from [Gretton et al., 2008]): Let $\mathbf{P}_{xy}$ be a Borel probability measure defined on a domain $\mathcal{X} \times \mathcal{Y}$, and let $\mathbf{P}_x$ and $\mathbf{P}_y$ be the respective marginal distributions on $\mathcal{X}$ and $\mathcal{Y}$. Given samples $Z = (X, Y) = \{(x_1, y_1), ..., (x_m, y_m)\}$ of size m drawn independently and identically distributed according to $\mathbf{P}_{xy}$, does $\mathbf{P}_{xy}$ factorise as $\mathbf{P}_{xy} = \mathbf{P}_x\mathbf{P}_y$? This corresponds to the question: Are $\mathbf{P}_x$ and $\mathbf{P}_y$ statistically independent? An independence test will distinguish between the hypothesises

$$H_0 : \mathbf{P}_{xy} = \mathbf{P}_x\mathbf{P}_y$$
$$H_1 : \mathbf{P}_{xy} \neq \mathbf{P}_x\mathbf{P}_y$$

As for two-sample-testing, it is desirable to have a statistic that is zero if and only if $\mathbf{P}_x$ and $\mathbf{P}_y$ are independent. The so-called *Hilbert Schmidt Independence Criterion* has this property. It is the squared Hilbert-Schmidt norm of the cross-covariance operator. We will now briefly describe where it comes from.

Let $\mathcal{F}$ be a RKHS with continuous feature mapping $\phi : \mathcal{X} \to \mathcal{F}$ and kernel $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ such that $\langle \phi(x)\phi(x') \rangle_\mathcal{F} = k(x, x')$; let $\mathcal{G}$ be another RKHS with continuous feature mapping $\psi : \mathcal{X} \to \mathcal{G}$ and kernel $l : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ such that $\langle \psi(y)\psi(y') \rangle_\mathcal{G} = k(y, y')$. The cross-covariance operator $C_{xy} : \mathcal{G} \to \mathcal{F}$ is defined such that for all $f \in \mathcal{F}$ and $g \in \mathcal{G}$

$$\langle f, C_{xy}g \rangle_\mathcal{F} = \mathbf{E}_{xy}([f(x) - \mathbf{E}_x(f(x))][g(y) - \mathbf{E}_y(g(y))]$$

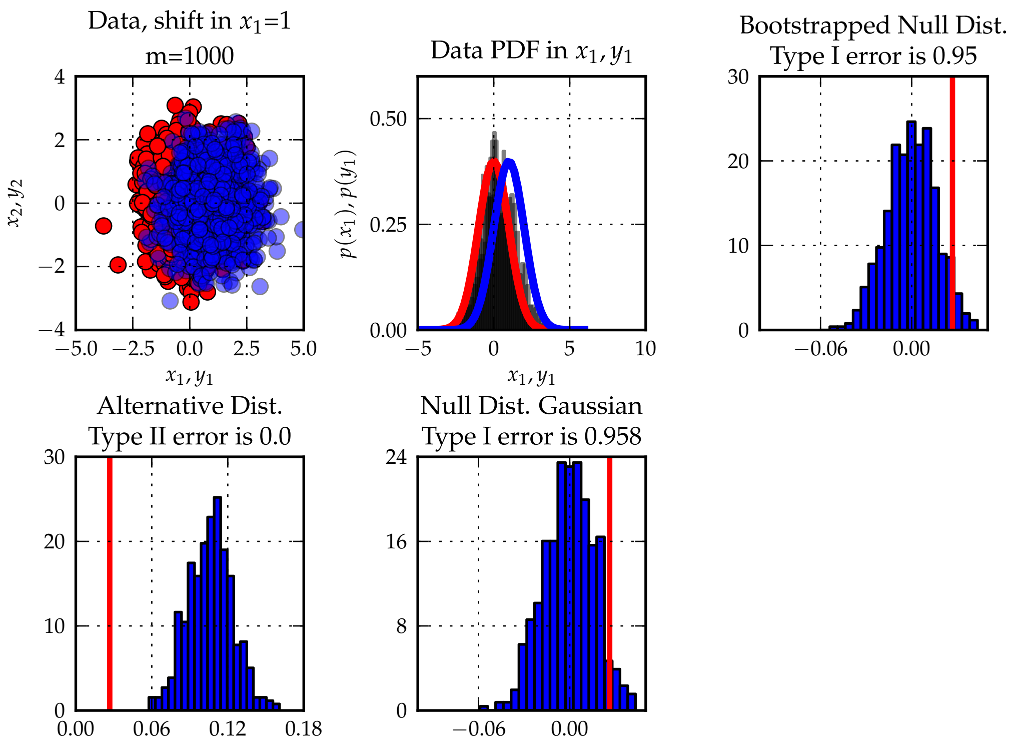


Figure 5.1: Screenshot of graphical python example for linear time MMD.

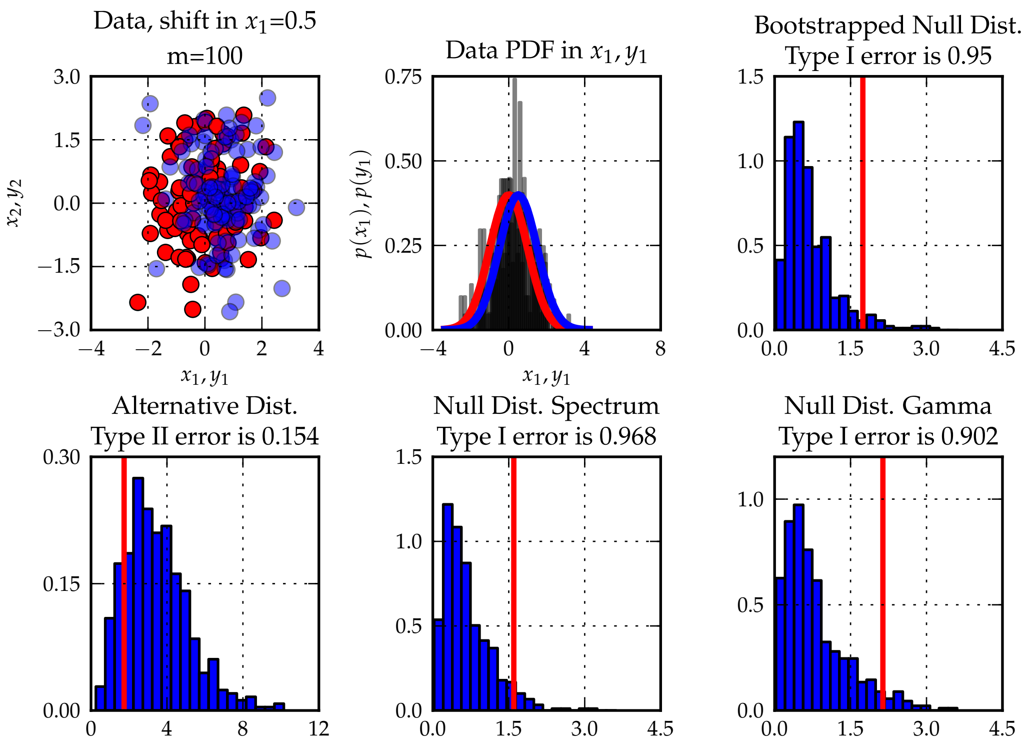


Figure 5.2: Screenshot of graphical python example for quadratic time MMD.

The operator itself can be written as

$$C_{xy} = \mathbf{E}_{xy}(\phi(x) - \mu_x) \otimes (\psi(y) - \mu_y)]$$

where $\otimes$ is the tensor product. This is the generalisation of the cross-covariance matrix between random vectors in a RKHS. When $\mathcal{F}$ and $\mathcal{G}$ are *universal*, then $||C_{xy}||$ is z is zero if and only if $H_0 : \mathbf{P}_{xy} = \mathbf{P}_x\mathbf{P}_y$ holds. Collecting everything gives the population expression for the HSIC. Let x', y' be independent copies of the random variables x, y respectively.

$$\begin{aligned}\text{HSIC}[\mathbf{P}_{xy}, \mathcal{F}, \mathcal{G}] = \mathbf{E}_{xx'yy'}[k(x,x')l(y,y')] + \mathbf{E}_{xx'}[k(x,x')]\mathbf{E}_{yy'}[l(y,y')] \\ - 2\mathbf{E}_{xy}[\mathbf{E}_{x'}[k(x,x')]\mathbf{E}_{y'}[l(y,y')]]\end{aligned} \tag{5.4}$$

See [Gretton et al., 2008] for details. SHOGUN implements a biased estimator of the HSIC along with various methods to approximate its null distribution.

5.3.1 Estimate of HSIC

We now describe the method to estimate the HSIC that is implemented in SHOGUN, as described in [Gretton et al., 2008, Equation 4]. All methods are implemented in `CHSIC`. The HSIC statistic has quadratic time and space costs, since it involves computing full kernel matrices and centring them.

A biased estimator for expression 5.4 is given by

$$\text{HSIC}_b[(X,Y), \mathcal{F}, \mathcal{G}] = \frac{1}{m^2}\text{trace}(\mathbf{KHLH})$$

where $\mathbf{K}, \mathbf{L}$ are the full kernel matrices of kernels k, l respectively and $\mathbf{H} = \mathbf{I} - \frac{1}{m}\mathbf{1}\mathbf{1}^T$ is a centring matrix with $\mathbf{1}$ being a $m \times m$ matrix of ones. In SHOGUN, this expression is not evaluated using matrix multiplication, but the centring is done by hand. Call `compute_statistic` in order to compute the estimate. Note that the method returns $m\,\text{HSIC}_b[(X,Y), \mathcal{F}, \mathcal{G}]$ since null distribution approximations work on m times null distribution.

Bootstrapping

As for any independence test in SHOGUN, bootstrapping can be used to approximate the null-distribution with any type of statistic. This results in a consistent, but slow test. Note that for each sample, the HSIC estimate has to be re-computed. The number of samples to take is the only parameter. As a rule of thumb, use at least 250 samples. See `bootstrap_null` in `CTwoDistributionsTestStatistic`, `CKernelIndependenceTestStatistic`, and `CHSIC`. Note that since the full kernel matrices have to be stored anyway when computing the HSIC estimate, in `bootstrap_null`, these are pre-computed automatically for the current bootstrapping instance. Bootstrapping is the only consistent HSIC test that is implemented in SHOGUN.

Gamma

Another, fast but heuristic method for approximating the null distribution for the HSIC is by matching the first two moments of a gamma distribution to it [Gretton et al., 2008, Equation 9]. This is not consistent but usually gives good results in practice. However, there are distributions which break the gamma test. Therefore, the type I error should always be monitored.

It uses

$$m\,\text{HSIC}_b(Z) \sim \frac{x^{\alpha-1}\exp(-\frac{x}{\beta})}{\beta^{\alpha}\Gamma(\alpha)} \tag{5.5}$$

where

$$\alpha = \frac{(\mathbf{E}(\text{HSIC}_b(Z)))^2}{\text{var}(\text{HSIC}_b(Z))} \qquad \text{and} \qquad \beta = \frac{m\,\text{var}(\text{HSIC}_b(Z))}{(\mathbf{E}(\text{HSIC}_b(Z)))^2}$$

Then, any threshold and p-value can be computed using the gamma distribution in expression 5.5. Computational costs are in $\mathcal{O}(m^2)$, similar for space.

To use that method for testing, use `set_null_approximation_method(HSIC_GAMMA)`, to be found in `CHSIC`.

Kernel Selection

Kernel selection for HSIC-based tests is an ongoing subject of research. In the near future, recent results on this topic will be implemented into SHOGUN. In the meantime, a good heuristic to start with, when using a Gaussian kernel as implemented in `CGaussianKernel`, is to use the median distance in the underlying data as kernel bandwidth. This way, the kernel captures at least the scaling of underlying data. In many cases, this leads to usable first results, however, the method may badly fail if signal is hidden at another length scale than the one of the overall data. The method is mentioned in [Gretton et al., 2012a, Appendix C].

In order to do this, note that SHOGUN's Gaussian kernel implementation uses a different parametrization as most other literature:

$$k(x, x') = \exp\left(\frac{||x - x'||_2^2}{\tau}\right)$$

where τ is the width that is passed to the constructor of `CGaussianKernel`. In order to translate a median distance d to this kernel, simply pass $\tau = d^2$.

SHOGUN implements classes that can compute median distances. Create an instance of the class `CEulideanDistance`, call the method `distance_matrix` to compute all pairwise distances of passed data. Then, use the method `matrix_median` of `CStatistics` in order to compute the median of all elements in the matrix. Store this distance and pass it to the Gaussian kernel as described above.

Note that the median is a very stable statistic, therefore, it is not necessary to compute all pairwise distances. A subset of a few hundred points is sufficient. This can be done via setting a subset to the instance of `CFeatures` that holds the data. To do so, create a random index permutation via calling method `randperm_vec` of class `CMath`, and only keep the n first indices where n is the number of distances that should be computed. Call `add_subset` on the instance of `CFeatures` that holds the data and give the indices as parameter. Then, this instance can be passed to `CEulideanDistance` as described above. Distances will only be computed for specified indices. Once the distance matrix is computed, call `remove_subset` to reset the original state of the data.

The whole procedure is included in all python examples, including the graphical ones. Note that since there is a kernel for each distribution, two kernel parameters have to be selected. Using the described median heuristic also leads to good results in some cases, as mentioned in [Gretton et al., 2008].

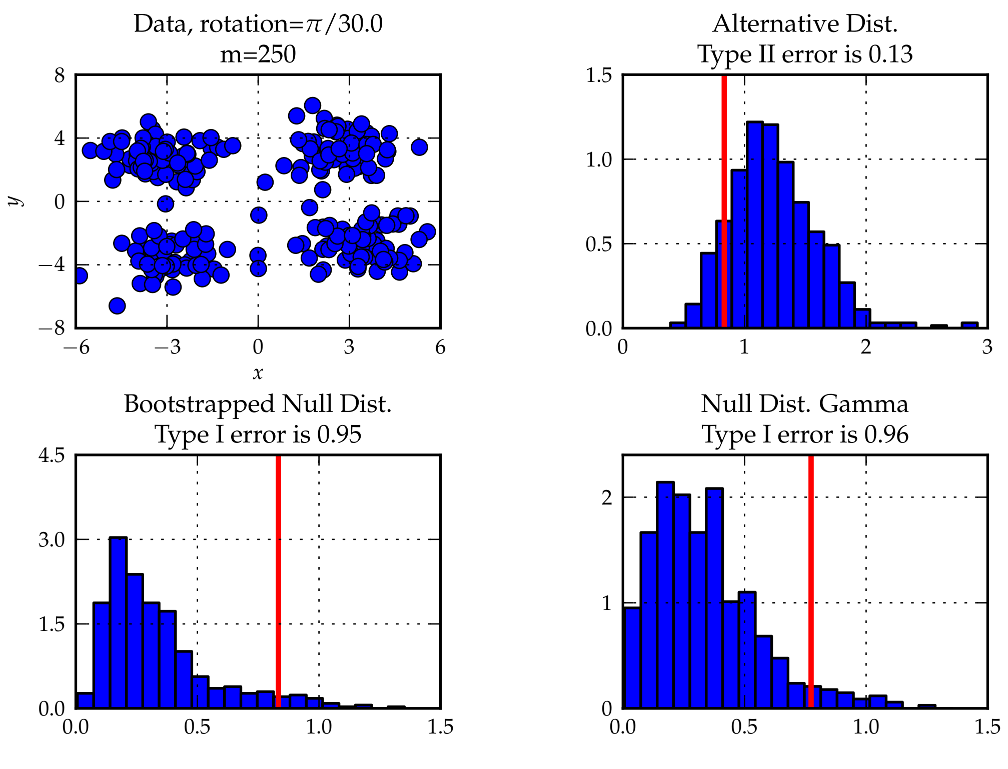


Figure 5.3: Screenshot of graphical python example for HSIC.

Example

There is a graphical python example which plots example data, alternative and null-distributions. See figure 5.3.

Chapter 6

Multitask learning

In this chapter we describe multitask learning algorithms available in the SHOGUN toolbox. In the toolbox we include descriptions of some multitask learning algorithms ported from two packages: SLEP (the Sparse LEarning Package) and the MALSAR (Multi-tAsk Learning via StructurAl Regularization) package.

6.1 L_1/L_q-norm regularized multitask learning

One of the simplest approaches to learn linear classification and regression models in the multitask environment is to come with regularization based on L_1/L_q norm of the common w hyperparameter

$$\|w\|_{1/q} = \sum_{t=1}^{T} \|w\|_q.$$

That kind of regularization in the same time pulls corresponding weights of hyperparameters w_t to be similar and pulls non-relevant feature weights to be zero.

6.1.1 Least squares linear regression

The algorithm learns a multitask linear least squares regression model of regression

$$f_t(x) = \langle w_t, x \rangle + b_t, \quad t = 1, \dots, T,$$

where T is a number of tasks, from the solution of the following optimization problem:

$$\min_{w} \sum_{t=1}^{T} \sum_{i \in G_t} (\langle w_t, x_i \rangle + b_t - y_i)^2 + \lambda \|w\|_{1/q},$$

where $G = \{G_1, \dots, G_T\}$ is a set of tasks' non-overlapping indices, $\forall_i x_i$ are feature vectors and $\forall_i y_i \in \mathbb{R}$ are labels.

The algoritm is implemented in `CMultitaskLeastSquaresRegression`.

6.1.2 Logistic regression

The algorithm learns a multitask linear logistic model of classification

$$f_t(x) = sign(\langle w_t, x\rangle + b_t), \quad t = 1, \dots, T,$$

where T is a number of tasks, from the solution of the following optimization problem:

$$\min_{w} \sum_{t=1}^{T} \sum_{i \in G_t} \frac{1}{|G_t|} \log(1 + \exp\left(-y_i(\langle w_t, x_i\rangle + b_t)\right) + \lambda \|w\|_{1/q},$$

where $G = \{G_1, \dots, G_T\}$ is a set of tasks' non-overlapping indices, $\forall_i x_i$ are feature vectors and $\forall_i y_i \in \{-1, 1\}$ are labels.

The algorithm is implemented in `CMultitaskLogisticRegression`.

6.2 Tree structured group lasso multitask learning

In some cases relations between tasks can be described via tree structure.

6.2.1 Least squares linear regression

The algorithm learns a multitask linear least squares regression model of regression

$$f_t(x) = \langle w_t, x\rangle + b_t, \quad t = 1, \dots, T,$$

where T is a number of tasks, from the solution of the following optimization problem:

$$\min_{w} \sum_{t=1}^{T} \sum_{i \in G_t} \left(\langle w_t, x_i\rangle + b_t - y_i\right)^2 + \lambda \|w\|_{1/q},$$

where $G = \{G_0, \dots, G_T\}$ is a set of tasks' tree indices, $\forall_i x_i$ are feature vectors and $\forall_i y_i \in \mathbb{R}$ are labels.

The algorithm is implemented in `CMultitaskLeastSquaresRegression`.

6.2.2 Logistic regression

The algorithm learns a multitask linear logistic model of classification

$$f_t(x) = sign(\langle w_t, x\rangle + b_t), \quad t = 1, \dots, T,$$

where T is a number of tasks, from the solution of the following optimization problem:

$$\min_{w,c} \sum_{t=1}^{T} \sum_{i \in G_t} \frac{1}{|G_t|} \log(1 + \exp\left(-y_i(\langle w_t, x_i\rangle + b_t)\right) + \lambda \|w\|_{1/q},$$

where $G = \{G_0, \dots, G_T\}$ is a set of tasks' tree indices, $\forall_i x_i$ are feature vectors and $\forall_i y_i \in \{-1, 1\}$ are labels.

The algorithm is implemented in `CMultitaskLogisticRegression`.

6.3 Low rank approximations

Tasks relationship can be constrained with models based on shared low-dimensional subspace. That can be done via solving the following optimization problem:

$$\min_{W=[w_1,\dots,w_T]} L(W) + \lambda\,\mathrm{rank}(W),$$

where L is a pre-defined loss function. It is known that the problem is NP-hard which makes it infeasible to solve in real applications. In practice similar problem is

$$\min_{W=[w_1,\dots,w_T]} L(W) + \lambda\|W\|_*,$$

where the sum of the singular values $\|W\|_* = \sum_i \sigma_i(W)$ is the trace norm.

6.3.1 Least squares linear regression

The algorithm learns a multitask linear least squares regression model of regression

$$f_t(x) = \langle w_t, x\rangle + b_t, \quad t = 1,\dots,T,$$

where T is a number of tasks, from the solution of the following optimization problem:

$$\min_{W=[w_1,\dots,w_T]} \sum_{t=1}^{T} \sum_{i\in G_t} (\langle w_t, x_i\rangle + b_t - y_i)^2 + \sum_i \sigma_i(W),$$

where $G = \{G_1,\dots,G_T\}$ is a set of tasks' non-overlapping indices, $\forall_i x_i$ are feature vectors and $\forall_i y_i \in \mathbb{R}$ are labels.

6.3.2 Logistic regression

The algorithm learns a multitask linear logistic model of classification

$$f_t(x) = sign(\langle w_t, x\rangle + b_t), \quad t = 1,\dots,T,$$

where T is a number of tasks, from the solution of the following optimization problem:

$$\min_{W=[w_1,\dots,w_T]} \sum_{t=1}^{T} \sum_{i\in G_t} \frac{1}{|G_t|} \log(1 + \exp(-y_i(\langle w_t, x_i\rangle + b_t)) + \sum_i \sigma_i(W),$$

where $G = \{G_1,\dots,G_T\}$ is a set of tasks' non-overlapping indices, $\forall_i x_i$ are feature vectors and $\forall_i y_i \in \{-1,1\}$ are labels.

The algorithm is implemented in `CMultitaskTraceLogisticRegression`.

6.4 Clustered multitask learning

Other approach assuming tasks may exhibit k-cluster structure. That kind of structure makes learned models of similar tasks (i.e. tasks of one cluster) to be closer to each other than to other tasks. The approach can be formalized to the following optimization problem

$$\min_{W=[w_1,\dots,w_T]} L(W) + \alpha(\mathrm{tr}\, W^T W - \mathrm{tr}\, F^T W^T W F) + \beta\, \mathrm{tr}\, W^T W,$$

where L is a pre-defined loss function. The problem can be also relaxed to be convex:

$$\min_{W=[w_1,\ldots,w_T]} L(W) + \rho_1 \eta (1+\eta)(\mathrm{tr}(W(\eta I + M)^{-1} W^T)$$

subject to $\mathrm{tr}\, M = k$, $M \preceq I$, $\eta = \frac{\rho_2}{\rho_1}$.

6.4.1 Least squares linear regression

The algorithm learns a multitask linear least squares regression model of regression

$$f_t(x) = \langle w_t, x \rangle + b_t, \quad t = 1, \ldots, T,$$

where T is a number of tasks, from the solution of the following optimization problem:

$$\min_{W=[w_1,\ldots,w_T]} \sum_{t=1}^{T} \sum_{i \in G_t} (\langle w_t, x_i \rangle + b_t - y_i)^2 + \rho_1 \eta (1+\eta)(\mathrm{tr}(W(\eta I + M)^{-1} W^T),$$

subject to $\mathrm{tr}\, M = k$, $M \preceq I$, $\eta = \frac{\rho_2}{\rho_1}$; where $G = \{G_1, \ldots, G_T\}$ is a set of tasks' non-overlapping indices, $\forall_i x_i$ are feature vectors and $\forall_i y_i \in \mathbb{R}$ are labels.

6.4.2 Logistic regression

The algorithm learns a multitask linear logistic model of classification

$$f_t(x) = sign(\langle w_t, x \rangle + b_t), \quad t = 1, \ldots, T,$$

where T is a number of tasks, from the solution of the following optimization problem:

$$\min_{W=[w_1,\ldots,w_T]} \sum_{t=1}^{T} \sum_{i \in G_t} \frac{1}{|G_t|} \log(1 + \exp\left(-y_i(\langle w_t, x_i \rangle + b_t)\right) + \rho_1 \eta (1+\eta)(\mathrm{tr}(W(\eta I + M)^{-1} W^T),$$

subject to $\mathrm{tr}\, M = k$, $M \preceq I$, $\eta = \frac{\rho_2}{\rho_1}$; where $G = \{G_1, \ldots, G_T\}$ is a set of tasks' non-overlapping indices, $\forall_i x_i$ are feature vectors and $\forall_i y_i \in \{-1, 1\}$ are labels.

The algorithm is implemented in `CMultitaskClusteredLogisticRegression`.

Appendix A

GNU Free Documentation License

License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an “aggregate” if the copyright resulting from the compilation is not used to limit the legal rights of the compilation’s users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document’s Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled “Acknowledgements”, “Dedications”, or “History”, the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to

11. RELICENSING

ADDENDUM: How to use this License for your documents

Bibliography

Allwein, E. L., Schapire, R. E., and Singer, Y. (2000). Reducing multiclass to binary: A unifying approach for margin classifiers. In *ICML*, pages 9–16.

Beygelzimer, A., Kakade, S., and Langford, J. (2006). Cover trees for nearest neighbor. In *ICML*, pages 97–104.

Devroye, L., Gyorfi, L., and Lugosi, G. (1996). *A probabilistic theory of pattern recognition*. Springer.

Dietterich, T. G. and Bakiri, G. (1995). Solving multiclass learning problems via error-correcting output codes. *J. Artif. Intell. Res. (JAIR)*, 2:263–286.

Escalera, S., Pujol, O., and Radeva, P. (2009). Separability of ternary codes for sparse designs of error-correcting output codes. *Pattern Recognition Letters*, 30(3):285–297.

Gretton, A., Borgwardt, K. M., Rasch, M. J., Schölkopf, B., and Smola, A. (2012a). A Kernel Two-Sample Test. *Journal of Machine Learning Research*, 13:671–721.

Gretton, A., Fukumizu, K., Harchaoui, Z., and Sriperumbudur, B. K. (2012b). A fast, consistent kernel two-sample test. pages 673—-681.

Gretton, A., Fukumizu, K., Teo, C., and Song, L. (2008). A kernel statistical test of independence.

Gretton, A., Sriperumbudur, B., Sejdinovic, D., Strathmann, H., Balakrishnan, S., Pontil, M., and Fukumizu, K. (2012c). Optimal kernel choice for large-scale two-sample tests. In *Advances in Neural Information Processing Systems*.

Hastie, T. and Tibshirani, R. (1997). Classification by pairwise coupling. In *NIPS*.

Ng, A. Y., Jordan, M. I., and Weiss, Y. (2001). On spectral clustering: Analysis and an algorithm. In *NIPS*, pages 849–856.

Rifkin, R. M. and Klautau, A. (2004). In defense of one-vs-all classification. *Journal of Machine Learning Research*, 5:101–141.

Sriperumbudur, B., Fukumizu, K., Gretton, A., Lanckriet, G. R. G., and Schölkopf, B. (2009). Kernel choice and classifiability for RKHS embeddings of probability distributions. In *Advances in Neural Information Processing Systems*.

Strathmann, H. (2012). M.Sc. Adaptive Large-Scale Kernel Two-Sample Testing.