Supplements for Multivariate Statistical Analysis

prepared by T.-Y. Li* supervised by H.-M. Zhang†

October 21st, 2020

Abstract

Now that the notes posted on https://zhuanlan.zhihu.com/p/91593024 are not exciting enough, it's encouraged to present some popular methods and techniques. In this supplementary material, we will cover topics listed below, emphasizing their intuitiveness without losing mathematical rigor. A number of related exercises are given at the end of the material to aid understanding. The references provide further exposition and will benefit those who wish to delve deeper.

Contents

References

[FLZZ] J. Fan, R. Li, C. Zhang, & H. Zou (2020). *Statistical Foundations of Data Science*. Chapman and Hall/CRC. https://orfe.princeton.edu/~jqfan/DataScience/

[HTF] T. Hastie, R. Tibshirani, & J. Friedman (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction* (2nd ed.). Springer. https://web.stanford.edu/~hastie/ElemStatLearn/

[SB] S. Shalev-Shwartz, & S. Ben-David (2014). *Understanding Machine Learning: From Theory to Algorithms*. CUP. https://www.cs.huji.ac.il/~shais/UnderstandingMachineLearning/

[BSS] A. S. Bandeira, A. Singer, & T. Strohmer (2020). *Mathematics of Data Science* (Draft: version 0.1). https://people.math.ethz.ch/~abandeira/BandeiraSingerStrohmer-MDS-draft.pdf

[B] L. Breiman (2001). "Statistical Modeling: The Two Cultures (with comments and a rejoinder by the author)", in *Statist. Sci.*. https://doi.org/10.1214/ss/1009213726

[JP] I. M. Johnstone, & D. Paul (2018). "PCA in High Dimensions: An Orientation", in *Proceedings of the IEEE*. https://doi.org/10.1109/JPROC.2018.2846730

[CFW] Z. Chen, J. Fan, & C. D. Wang (2020). "High-dimensional factor model and its applications to statistical machine learning (in Chinese)", in *Sci. Sin. Math.*. https://doi.org/10.1360/SSM-2020-0041

*E-Mail: kellty@pku.edu.cn
†Homepage: http://scholar.pku.edu.cn/zhanghuiming

1 Classification

The problem of predicting a *discrete* indicator Y from an observation vector $\boldsymbol{X} = \boldsymbol{x}$ is called **classification**, a.k.a. discrimination or pattern recognition. Often, the covariates x_j in $\boldsymbol{x} = (x_1, \ldots, x_p)'$ are called features. Classification is one of the fundamental problems in *supervised learning*. Assume that we are given i.i.d. data $(\boldsymbol{X}_1, Y_1), \ldots, (\boldsymbol{X}_n, Y_n)$, where $\boldsymbol{X}_i$'s take values in $\mathcal{X} = \mathbb{R}^p$ and Y_i's take values in a finite set $\mathcal{Y}$ consisting of *labels*. When there are G classes, one may put $\mathcal{Y} = \{1, 2, \cdots, G\}$ for convenience.

1.1 Support Vector Machines

To begin with, we describe **separating hyperplane** classifiers. These procedures, including Fisher's linear discriminant analysis and logistic regression, construct linear decision boundaries that explicitly try to separate the data into different classes as well as possible. A hyperplane is defined by

$$\Pi_{\alpha,\boldsymbol{\beta}} = \{\boldsymbol{x} \in \mathbb{R}^p : \alpha + \boldsymbol{x}'\boldsymbol{\beta} = 0\},$$

where $\alpha \in \mathbb{R}$ and $\boldsymbol{\beta} \in \mathbb{R}^p$ is a unit vector, i.e., $\|\boldsymbol{\beta}\|_2 = \sqrt{\boldsymbol{\beta}'\boldsymbol{\beta}} = 1$.

Consider binary classification, where the class labels are coded as ± 1. The *optimal* separating hyperplane separates the two classes and maximizes the distance to the closest point from either class.

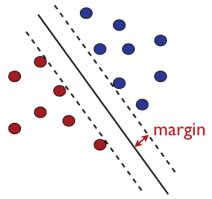
We say that the training data are **linearly separable**, if there exists a hyperplane $\Pi_{\alpha,\boldsymbol{\beta}}$ such that

$$(\alpha + \boldsymbol{X}_i'\boldsymbol{\beta})Y_i > 0, \quad \forall 1 \le i \le n,$$

which means $\Pi_{\alpha,\boldsymbol{\beta}}$ is a perfect classifier. In the linearly separable case, the distance[i)] from $\boldsymbol{X}_i$ to $\Pi_{\alpha,\boldsymbol{\beta}}$ is given by

$$d_i = d(\boldsymbol{X}_i, \Pi_{\alpha,\boldsymbol{\beta}}) = \frac{|\alpha + \boldsymbol{X}_i'\boldsymbol{\beta}|}{\|\boldsymbol{\beta}\|_2} = (\alpha + \boldsymbol{X}_i'\boldsymbol{\beta})Y_i.$$

The **support vector machine** (SVM) has proven to be one of the most successful classification tools, whose idea is basically



maximizing the **margin**

$$M = \min_{1 \le i \le n} (\alpha + \boldsymbol{X}_i'\boldsymbol{\beta})Y_i \quad \text{w.r.t. } (\alpha, \boldsymbol{\beta}) \in \mathbb{R} \times \mathbb{S}^{p-1},$$

where $\mathbb{S}^{p-1}$ denotes the unit sphere in $\mathbb{R}^p$. A point $\boldsymbol{X}_i$ is called the *support vector* if it attains the margin, i.e., $d_i = M$.

In applications the training data are usually not linearly separable. Then the SVM allows some points to be on the wrong side of the hyperplane. This leads to the general SVM problem:

$$\max \ M \quad \text{w.r.t. } (\alpha, \boldsymbol{\beta}) \in \mathbb{R} \times \mathbb{S}^{p-1}$$

$$\text{s.t.} \begin{cases} (\alpha + \boldsymbol{X}_i'\boldsymbol{\beta})Y_i \ge (1 - \xi_i)M, \\ \xi_i \ge 0, \quad \sum \xi_i \le B, \end{cases} \quad i = 1, \cdots, n,$$

where ξ_i's are *slack* variables, and $B > 0$ is a pre-specified *tuning* parameter.

The general SVM is sometimes called Soft-SVM, and the original one ($B = 0$) may be called Hard-SVM. Once the optimal parameter $(\hat{\alpha}, \hat{\boldsymbol{\beta}}) \in \mathbb{R} \times \mathbb{S}^{p-1}$ is solved, the fitted SVM will allocate a new observation $\boldsymbol{x}$ to the class labeled by

$$\hat{g}(\boldsymbol{x}) = \text{sign}(\hat{f}(\boldsymbol{x})) \text{ with } \hat{f}(\boldsymbol{x}) = \hat{\alpha} + \boldsymbol{x}'\hat{\boldsymbol{\beta}}.$$

By the Lagrange multiplier method and Exercise 1, it can be seen that $\text{sign}(\hat{\alpha} + \boldsymbol{x}'\hat{\boldsymbol{\beta}}) = \text{sign}(\tilde{\alpha} + \boldsymbol{x}'\tilde{\boldsymbol{\beta}})$ for all $\boldsymbol{x} \in \mathbb{R}^p$, where $(\tilde{\alpha}, \tilde{\boldsymbol{\beta}}) \in \mathbb{R} \times \mathbb{R}^p$ is obtained by *minimizing the regularized loss*

$$\frac{1}{n}\sum_{i=1}^{n} [1 - (\alpha + \boldsymbol{X}_i'\boldsymbol{\beta})Y_i]_+ + \lambda \|\boldsymbol{\beta}\|_2^2$$

for a multiplier $\lambda = \lambda(B) > 0$. Here $L^{\text{hinge}}(\hat{y}, y) = (1 - \hat{y}y)_+ = \max\{0, 1 - \hat{y}y\}$ is called the *hinge loss*.

[i)] cf. https://math.stackexchange.com/q/1210545

In light of this, a **large margin classifier** is defined by $\widehat{g}(\boldsymbol{x}) = \text{sign}(\widehat{f}(\boldsymbol{x}))$ with

$$\widehat{f} \in \underset{f \in \mathcal{H}_K}{\arg\min} \left\{ \frac{1}{n} \sum_{i=1}^{n} \phi(f(\boldsymbol{X}_i) Y_i) + \lambda \|f\|_{\mathcal{H}_K}^2 \right\},$$

where $\mathcal{H}_K$ is the Hilbert space with a reproducing kernel[ii] $K(\cdot, \cdot)$, ϕ is a *Fisher consistent* loss function (Exercise 2), and $\lambda > 0$ serves as the tuning parameter. The **kernel SVM** is a special case when we choose $\phi(f(\boldsymbol{X}_i) Y_i) = L^{\text{hinge}}(f(\boldsymbol{X}_i), Y_i)$. The theory of *reproducing kernel Hilbert spaces* guarantees that the large margin classifier has a finite dimensional expression in terms of the reproducing kernel function. That is,

$$\widehat{f}(\boldsymbol{x}) = \widehat{\alpha} + \sum_{i=1}^{n} \widehat{\beta}_i K(\boldsymbol{x}, \boldsymbol{X}_i)$$

with

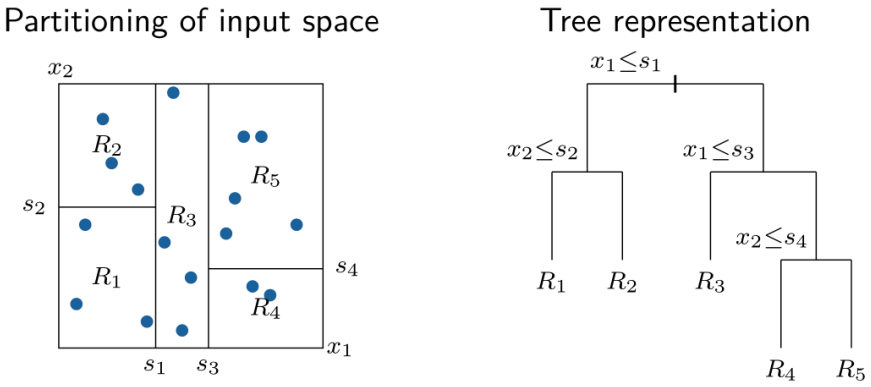
$$(\widehat{\alpha}, \widehat{\boldsymbol{\beta}}) \in \underset{(\alpha, \boldsymbol{\beta}) \in \mathbb{R} \times \mathbb{R}^n}{\arg\min} \left\{ \frac{1}{n} \sum_{i=1}^{n} \phi\left(\left(\alpha + \sum_{j=1}^{n} \beta_j K(\boldsymbol{X}_i, \boldsymbol{X}_j) \right) Y_i \right) + \lambda \boldsymbol{\beta}' \mathbf{K} \boldsymbol{\beta} \right\},$$

where $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_n)'$ and $\mathbf{K} = (K(\boldsymbol{X}_i, \boldsymbol{X}_j))_{1 \le i,j \le n}$.

Another extension of SVM is taking sparsity into account. To that end, the penalty $\|\boldsymbol{\beta}\|_2^2$ may be replaced with $\|\boldsymbol{\beta}\|_1 = \sum |\beta_j|$ to yield the ℓ^1 SVM, which reminds us of ridge regression and the LASSO.

1.2 Trees and Random Forests

A *tree*-structured classifier is typically constructed by **recursive partitioning** algorithms in the input space, which can be graphically presented by a **decision tree** whose terminals correspond to the final



partitioned regions $R_1, \ldots, R_M$. The classifier makes a constant prediction within each region by using a **majority vote**. Thus, if a new observation $\boldsymbol{x}$ is put into the region R_m by the decision tree, then the predicted label should be

$$T(\boldsymbol{x}) = \hat{y}(m) = \underset{y \in \mathcal{Y}}{\arg\max} \sum_{i: \boldsymbol{X}_i \in R_m} \mathbb{1}_{[Y_i = y]}, \qquad m = 1, \cdots, M.$$

The key challenge in this strategy is to find a good partition. Even if we restrict our attention to seemingly simple regions, finding an optimal partition to minimize the training error is computationally infeasible. Practically, we use a "greedy" approach referred to as **recursive binary splitting**:

1. Start with the trivial partition $\{R_1\} = \{\mathcal{X}\}$;
2. On a region R_m, for variable j and split point s (that are chosen to minimize some *impurity function*) define two new regions

   $$R_{\le}(j, s) = \{\boldsymbol{x} = (x_1, \ldots, x_p)' \in R_m : x_j \le s\} \quad \& \quad R_{>}(j, s) = \{\boldsymbol{x} = (x_1, \ldots, x_p)' \in R_m : x_j > s\}$$

   and thence replacing R_m with $R_{\le}(j, s)$ and $R_{>}(j, s)$ gives a new partition;

[ii] cf. http://www.stat.cmu.edu/~ryantibs/statml/lectures/RKHSnotes.pdf & http://www.gatsby.ucl.ac.uk/~gretton/coursefiles/Slides5A.pdf

3. Repeat splitting process in step 2 on each of the resulting regions until some stopping criterion is met (e.g., no region contains more than 5 training data points).

Note that in step 2 if x_j is a continuous variable, we search over all s in a discrete set (e.g., set of midpoints between observed values or some quantiles of the empirical distribution). The impurity of a tree classifier T is measured by $\sum_{m=1}^{|T|} N_m Q_m$, where $|T|$ is the number of partitioned regions R_m's that correspond to T uniquely, $N_m = \#\{i : \boldsymbol{X}_i \in R_m\}$, and Q_m is one of the following **impurity functions** (Exercise 3) of R_m

- *Misclassification error*: $1 - \hat{p}_{m\hat{y}(m)} = 1 - \max_{y\in\mathcal{Y}} \hat{p}_{my}$
- *Gini index*: $\sum_{y\neq y'} \hat{p}_{my}\hat{p}_{my'} = \sum_{y\in\mathcal{Y}} \hat{p}_{my}(1-\hat{p}_{my}) = 1 - \sum_{y\in\mathcal{Y}} \hat{p}_{my}^2$
- *Cross-entropy* or *deviance*: $-\sum_{y\in\mathcal{Y}} \hat{p}_{my} \log \hat{p}_{my}$

where $\hat{p}_{my} = \frac{1}{N_m} \sum_{i:\boldsymbol{X}_i\in R_m} \mathbb{1}_{[Y_i=y]}$ denotes the observed proportion of class y in region R_m.

In order to avoid overfitting, one cuts back the tree ("**pruning**") after running the above algorithm. This is done by minimizing the *cost-complexity* criterion

$$C_\alpha(T) = \sum_{m=1}^{|T|} N_m Q_m + \alpha|T|,$$

where the tuning parameter $\alpha \geq 0$ may be chosen by cross-validation. Note that a subtree obtained by pruning the original tree has some internal (non-terminal) node(s) collapsed. Often, one uses the Gini index for partition building and the misclassification error for pruning.

The flexibility of classification trees is decided by the tree depth. To obtain a small bias the tree needs to be grown deep, but this results in a high variance. Apart from pruning, one may appeal to *ensemble methods* for improving the performance. The damage due to instability of a single predictor can be mitigated by combining multiple predictors to form a final prediction. For mathematical intuition, see Exercise 4.

Bootstrap **agg**regat**ing** (Bagging) was proposed by Breiman as a general technique for improving unstable predictive algorithms. Denote the training data by $Z_n = \{(Y_i, \boldsymbol{X}_i) : 1 \leq i \leq n\}$. Consider a learning algorithm g that takes Z_n as its input and produces a predictor g_{Z_n}. For classification problems, g_{Z_n} is the fitted classifier, and the **bagging** estimator is defined as the outcome of the *majority vote* of several bootstrap classifiers, i.e.,

$$\hat{g}^{\text{bag}}(\boldsymbol{x}) = \arg\max_{y\in\mathcal{Y}} \sum_{b=1}^{B} \mathbb{1}_{\left[g_{Z_n^{(*b)}}(\boldsymbol{x})=y\right]}, \quad \boldsymbol{x} \in \mathcal{X},$$

where $Z_n^{(*b)} = \{(Y_i^{(*b)}, \boldsymbol{X}_i^{(*b)}) : 1 \leq i \leq n\}$ is a **bootstrap**[iii] sample drawn by sampling from the training data Z_n *with replacement*. Note that bagging uses a randomized method, namely bootstrap, to generate a collection of classification trees.

Random forests put another level of randomization on top of bagging by using a randomly selected subset of features to construct each classification tree. The idea of using randomly selected variables aims to reduce the correlations between the trees, which can further enhance the variance reduction capability of bagging. The following algorithm shows how to grow a random forest.

1. For $b = 1$ to B (*can run in parallel*):
   i. Draw a bootstrap sample $Z_n^{(*b)}$ from the training data Z_n.
   ii. Apply a recursive partition method to $Z_n^{(*b)}$ to build a classification tree T_b^*. Before splitting each node, randomly select q out of the total of p input features. For the stopping criterion, a node becomes a terminal when its corresponding region contains only observations of the same class or no more than $N_{\min}$ data point(s).
   iii. Save the classification tree T_b^*.
2. The random forest classifier is given by $\hat{g}^{\text{RF}}(\boldsymbol{x}) = \arg\max_{y\in\mathcal{Y}} \sum_{b=1}^{B} \mathbb{1}_{\left[T_b^*(\boldsymbol{x})=y\right]}, \quad \boldsymbol{x} \in \mathcal{X}.$

Here the default values of tuning parameters are suggested to be $q = \lfloor\sqrt{p}\rfloor$ and $N_{\min} = 1$.

[iii] cf. http://www.stat.cmu.edu/~larry/=sml/Boot.pdf

2 Cluster Analysis

Cluster analysis is a learning task where class labels are *unobservable*, and we aim to find groups in the observed data $\{\boldsymbol{x}_i\}_{i=1}^n \subset \mathbb{R}^p$. It is an important example of the so-called *unsupervised learning*. We seek to partition $\boldsymbol{x}_i$'s into distinct groups, called **clusters**, so that the observations within each cluster are similar to each other, while observations in different clusters are quite different from each other. Cluster analysis can be useful for summarizing data and discovering new insights.

A **clustering** is a function C that assigns each observation $\boldsymbol{x}_i$ to a cluster $k \in \{1, \cdots, K\}$, where the number K of potential clusters is to be discussed later. Given pairwise *dissimilarities* $D(\boldsymbol{x}_i, \boldsymbol{x}_j)$ between observations, e.g., $D(\boldsymbol{x}_i, \boldsymbol{x}_j) = \|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2^2$, the **within-cluster scatter** is defined as

$$W_K(C) = \frac{1}{2} \sum_{k=1}^{K} \frac{1}{n_k(C)} \sum_{i,j:C(\boldsymbol{x}_i)=C(\boldsymbol{x}_j)=k} D(\boldsymbol{x}_i, \boldsymbol{x}_j),$$

where $n_k(C) = \#\{i : C(\boldsymbol{x}_i) = k\}$. A smaller $W_K(C)$ is undoubtedly better. However, minimizing W_K over all possible clusterings C is computationally costly. So we'll have to settle for an *approximation*.

2.1 K-Means Clustering

One of the oldest approaches to clustering is to find K representative points, called *prototypes* or *cluster centers*, and then divide the data into groups based on which prototype they are closest to.

From now on, let $D(\boldsymbol{x}_i, \boldsymbol{x}_j) = \|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2^2$. Then it can be shown that (Exercise 5)

$$W_K(C) = \sum_{k=1}^{K} \sum_{i:C(\boldsymbol{x}_i)=k} \|\boldsymbol{x}_i - \hat{\boldsymbol{\mu}}_k\|_2^2 = \min_{\boldsymbol{\mu}_1,\ldots,\boldsymbol{\mu}_K \in \mathbb{R}^p} \sum_{k=1}^{K} \sum_{i:C(\boldsymbol{x}_i)=k} \|\boldsymbol{x}_i - \boldsymbol{\mu}_k\|_2^2,$$

where $\hat{\boldsymbol{\mu}}_k = \frac{1}{n_k(C)} \sum_{i:C(\boldsymbol{x}_i)=k} \boldsymbol{x}_i$, and this $W_K(C)$ is often called **within-group sum of squares** (WGSS).

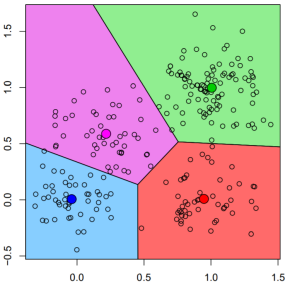
Hence our problem is the same as minimizing the enlarged criterion

$$\sum_{k=1}^{K} \sum_{i:C(\boldsymbol{x}_i)=k} \|\boldsymbol{x}_i - \boldsymbol{\mu}_k\|_2^2$$

over both clusterings C and $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K \in \mathbb{R}^p$. To solve it, **$K$-means clustering** (Exercise 6), also referred to as **Lloyd's algorithm**, iteratively updates the centers of the clusters and the cluster memberships:

1. Randomly choose K centroids $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K$ as starting values.
   A typical choice is to pick K points from the data $\{\boldsymbol{x}_i\}_{i=1}^n$.
2. Minimize over C: Cluster each observation $\boldsymbol{x}_i$ to the cluster $C(\boldsymbol{x}_i) = \arg\min_k \|\boldsymbol{x}_i - \boldsymbol{\mu}_k\|_2$.
3. Minimize over $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K$: Replace the centroids $\boldsymbol{\mu}_k$ with the sample mean within the k^{th} cluster.
4. Iterate steps 2 and 3 until the clustering results do not change.
5. Try multiple initializations, and output the solution with the smallest WGSS.

One can see that the WGSS is *non-increasing* in each iteration (check!) and the clustering will converge to a *local* minimum very fast.



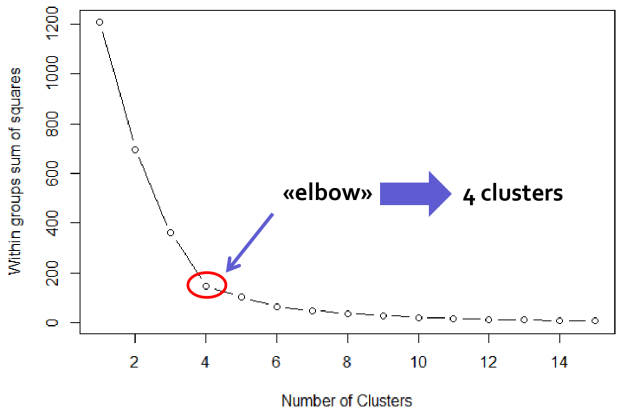
This yields a *Voronoi tessellation*[iv] of $\mathbb{R}^p$. Given $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K \in \mathbb{R}^p$, the **Voronoi sets** V_k, $k = 1, \cdots, K$, defined by

$$V_k = \left\{ \boldsymbol{x} \in \mathbb{R}^p : \|\boldsymbol{x} - \boldsymbol{\mu}_k\|_2^2 \le \|\boldsymbol{x} - \boldsymbol{\mu}_{k'}\|_2^2, \ \forall k' = 1, \cdots, K \right\}$$

are convex polyhedra covering $\mathbb{R}^p$.

In most exploratory applications, the number K is unknown and its choice is a hard problem, especially for explanation. The WGSS W_K measures how tightly grouped the clusters are, so we may plot W_K against K and hope to see a clear "elbow".

[iv] cf. https://www.r-bloggers.com/2015/02/k-means-clustering-and-voronoi-sets/



There are a number of statistics to make this eyeballing approach more formal. For example, Calinski and Harabasz proposed $K^*_{\mathrm{CH}} = \arg\max_K \mathrm{CH}(K)$ with the **CH index**

$$\mathrm{CH}(K) = \frac{B_K/(K-1)}{W_K/(n-K)},$$

where the **between-group sum of squares** (BGSS)

$$B_K = \sum_{k=1}^{K} n_k(C)\,\|\hat{\boldsymbol{\mu}}_k - \overline{\boldsymbol{x}}\|_2^2 = \sum_{i=1}^{n} \|\boldsymbol{x}_i - \overline{\boldsymbol{x}}\|_2^2 - W_K$$

measures how spread apart the clusters are from each other, where $\overline{\boldsymbol{x}} = \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{x}_i$ is the overall average.

3 High Dimensionality

In the information era, *big data* are ubiquitous. Accompanying big data is the rise of *dimensionality*, which means that the dimension p of covariates becomes relatively large relative to the sample size n, and high dimensionality characterizes many contemporary statistical problems. The main goals of high-dimensional statistical analysis are

- to construct a method as effective as possible to predict future observations and
- to gain insight into the relationship between features and responses for scientific purposes.

High dimensionality has a significant impact on computation, spurious correlation, noise accumulation, and theoretical studies. It's almost impossible to deliver a comprehensive overview in a brief introduction. Thus, in what follows, we shall only address some selected illuminating issues.

3.1 High-Dimensional PCA

Data is often represented as a matrix, for which reason linear algebra is one of the key tools in data analysis. Perhaps more surprising is the fact that *spectral* properties of data matrices play a crucial role. We will illustrate this importance with a discussion of tools from *random matrix theory*[v] to better understand the performance of PCA in the high-dimensional regime.

When faced with a high-dimensional dataset, a natural approach is to *reduce its dimension*, either by projecting it to a lower-dimensional space or by finding a better representation of the data using a small number of meaningful features. Beyond data compression and visualization, dimension reduction facilitates downstream analysis such as clustering and regression that perform significantly better in lower dimensions.

Based on the above ideas, PCA is a useful tool for summarizing high-dimensional data. However, it has an obvious drawback that each principal component uses all variables. This not only requires a high signal-to-noise ratio, but also makes principal components hard to interpret.

[v] cf. http://people.mpim-bonn.mpg.de/gborot/files/Mongolia-25sept2015.pdf & https://saasweb.hku.hk/workshop/rmcdaw2019/

Note that the usual PCA exploits the eigen-decomposition of the sample covariance matrix. When the dimensionality p is high, the sample covariance matrix is no longer a good estimator for the true covariance matrix, and hence the usual PCA may have poor performance. In fact, we may encounter the *inconsistency* phenomenon, which holds generally for the *spike covariance model*.

Let's consider a particularly simple, yet relevant, example where the population covariance matrix is an identity with a rank-1 perturbation. To be explicit, suppose that $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_n$ are independently drawn from $\boldsymbol{G} + \sqrt{\beta} G_0 \boldsymbol{u}$, where $\boldsymbol{G} \sim \mathcal{N}_p(\boldsymbol{0}_p, \boldsymbol{I}_p)$ and $G_0 \sim \mathcal{N}(0,1)$ are independent, $\beta > 0$ and $\boldsymbol{u} \in \mathbb{S}^{p-1}$. That is, $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_n \overset{\text{i.i.d.}}{\sim} \mathcal{N}_p(\boldsymbol{0}_p, \boldsymbol{\Sigma})$ with

$$\boldsymbol{\Sigma} = \boldsymbol{I}_p + \beta \boldsymbol{u}\boldsymbol{u}',$$

which we refer to as a **rank-1 spike model**. The noise part $\boldsymbol{G}$ is high-dimensional and isotropic, whereas the signal part $\sqrt{\beta} G_0 \boldsymbol{u}$ resides on a line in the direction of $\boldsymbol{u}$. We can regard β as the ratio of the signal variance (in the $\boldsymbol{u}$-direction) to the noise variance (in each direction).

Now that there exists some linear low-dimensional structure in the data, can we expect to capture it with PCA? For simplicity we will try to understand the spectral properties of

$$\boldsymbol{S}_n = \frac{1}{n} \sum_{i=1}^n \boldsymbol{X}_i \boldsymbol{X}_i'.$$

Let $\boldsymbol{Z}_i = \boldsymbol{\Sigma}^{-1/2} \boldsymbol{X}_i \overset{\text{i.i.d.}}{\sim} \mathcal{N}_p(\boldsymbol{0}_p, \boldsymbol{I}_p)$ and $\boldsymbol{W}_n = \frac{1}{n} \sum_{i=1}^n \boldsymbol{Z}_i \boldsymbol{Z}_i' \sim W_p(\boldsymbol{I}_p/n, n)$, then (check!)

$$\boldsymbol{S}_n = \boldsymbol{\Sigma}^{1/2} \boldsymbol{W}_n \boldsymbol{\Sigma}^{1/2} \text{ has the same eigenvalues as } \boldsymbol{W}_n \boldsymbol{\Sigma}.$$

Denote by $\lambda_j(\bigstar)$ the j^{th} largest eigenvalue of a Hermitian matrix $\bigstar$. To find $\lambda_1(\boldsymbol{S}_n) = \lambda_1(\boldsymbol{W}_n \boldsymbol{\Sigma})$, let $\boldsymbol{w} \in \mathbb{S}^{p-1}$ be the corresponding eigenvector such that

$$\boldsymbol{W}_n(\boldsymbol{I}_p + \beta \boldsymbol{u}\boldsymbol{u}')\boldsymbol{w} = \lambda_1(\boldsymbol{S}_n)\boldsymbol{w}, \quad \text{or} \quad (\lambda_1(\boldsymbol{S}_n)\boldsymbol{I}_p - \boldsymbol{W}_n)\boldsymbol{w} = \beta \boldsymbol{W}_n \boldsymbol{u}\boldsymbol{u}'\boldsymbol{w}.$$

Intuitively (a formal justification is possible), $\lambda_1(\boldsymbol{S}_n) > \lambda_1(\boldsymbol{W}_n)$, so

$$\boldsymbol{w} = \beta(\lambda_1(\boldsymbol{S}_n)\boldsymbol{I}_p - \boldsymbol{W}_n)^{-1} \boldsymbol{W}_n \boldsymbol{u}\boldsymbol{u}'\boldsymbol{w}.$$

Note that $\boldsymbol{u}'\boldsymbol{w} \neq 0$, for otherwise $\boldsymbol{w} = \boldsymbol{0}_p$. Premultiplying by $\boldsymbol{u}'$ and dividing by $\beta \boldsymbol{u}'\boldsymbol{w}$ lead to

$$\frac{1}{\beta} = \boldsymbol{u}'(\lambda_1(\boldsymbol{S}_n)\boldsymbol{I}_p - \boldsymbol{W}_n)^{-1} \boldsymbol{W}_n \boldsymbol{u}.$$

Write the eigen-decomposition of $\boldsymbol{W}_n$ as $\sum_{j=1}^p \lambda_j(\boldsymbol{W}_n) \boldsymbol{v}_j \boldsymbol{v}_j'$, where $\boldsymbol{v}_j$'s are orthonormal eigenvectors. It follows that

$$\frac{1}{\beta} = \sum_{j=1}^p \frac{\lambda_j(\boldsymbol{W}_n)}{\lambda_1(\boldsymbol{S}_n) - \lambda_j(\boldsymbol{W}_n)} (\boldsymbol{v}_j'\boldsymbol{u})^2.$$

By symmetry, $\mathbb{E}(\boldsymbol{v}_j'\boldsymbol{u})^2 = \frac{1}{p} \|\boldsymbol{u}\|_2^2 = \frac{1}{p}$, which implies (can be made rigorous)

$$\frac{1}{\beta} = \lim_{p \to \infty} \frac{1}{p} \sum_{j=1}^p \frac{\lambda_j(\boldsymbol{W}_n)}{\lambda_1(\boldsymbol{S}_n) - \lambda_j(\boldsymbol{W}_n)}.$$

Suppose that $p/n \to \gamma \in (0,1)$, then the **Marčenko-Pastur theorem**[vi] asserts that the *empirical spectral distribution* $\frac{1}{p} \sum_{j=1}^p \delta_{\lambda_j(\boldsymbol{W}_n)}$, where $\delta_\bullet$ stands for the Dirac mass, converges weakly to

$$\mathrm{d}F_\gamma(\lambda) = \frac{1}{2\pi} \frac{\sqrt{(\gamma_+ - \lambda)(\lambda - \gamma_-)}}{\gamma \lambda} \mathbb{1}_{[\gamma_-, \gamma_+]}(\lambda) \, \mathrm{d}\lambda,$$

where $\gamma_\pm = (1 \pm \sqrt{\gamma})^2$. We thus have for $\lim \lambda_1(\boldsymbol{S}_n) = \lambda > \gamma_+$ that (Exercise 7)

$$\frac{1}{\beta} = \int \frac{t}{\lambda - t} \, \mathrm{d}F_\gamma(t) = \frac{1}{4\gamma} \left[2\lambda - (\gamma_- + \gamma_+) - 2\sqrt{(\lambda - \gamma_-)(\lambda - \gamma_+)} \right].$$

[vi] cf. http://galton.uchicago.edu/~lalley/Courses/383/Wigner.pdf
& http://www.math.wisc.edu/~valko/courses/833/2009f/lec_6_7.pdf
Understanding the distribution of eigenvalues of random matrices is in the core of random matrix theory.

The case $\lambda = \gamma_+$ gives the *critical* signal-to-noise ratio

$$\beta_c = 4\gamma/(\gamma_+ - \gamma_-) = \sqrt{\gamma}.$$

Roughly speaking, one can observe the pop out of $\lambda_1(\boldsymbol{S}_n)$ from the Marčenko-Pastur distribution ($\beta = 0$) when and only when $\beta > \sqrt{p/n}$. This phenomenon is known as the **BBP phase transition**, which is named after Baik, Ben Arous, and Péché.

Low signal-to-noise ratio, small sample size, and high dimensionality are all obstacles for detecting linear structures in noisy high-dimensional data. Assume that $p/n \to \gamma > 0$. Let $\hat{\boldsymbol{u}} \in \mathbb{S}^{p-1}$ be the first eigenvector of $\boldsymbol{S}_n$. It can be shown that

$$|\boldsymbol{u}'\hat{\boldsymbol{u}}|^2 \xrightarrow{\text{a.s.}} (\beta^2 - \gamma)_+/(\beta^2 + \gamma\beta) < 1.$$

Also, $\lim |\boldsymbol{u}'\hat{\boldsymbol{u}}| > 0$ if and only if $\beta > \sqrt{\gamma}$.

It is possible to obtain a consistent estimator when the estimated eigenvector is sparse. **Sparse PCA** is a method for dimensionality reduction that *selects a subset of variables* to present the principal components. First, we should mention that the simple thresholding of the loadings of principal components can lead to sparse PCA, though the method can be improved.

A direct way to select variables is by adding an ℓ^1 constraint to the loadings. This approach was considered by Jolliffe, Trendafilov, and Uddin, who proposed **SCoTLASS** to derive the sparse principal components:

$$\max_{\boldsymbol{\xi}_k \in \mathbb{S}^{p-1}} \boldsymbol{\xi}_k' \boldsymbol{S} \boldsymbol{\xi}_k \quad \text{s.t.} \quad \boldsymbol{\xi}_j' \boldsymbol{\xi}_k = 0,\ j < k;\ \&\ \|\boldsymbol{\xi}_k\|_1 \le t$$

for some tuning parameter $t > 0$, where $\boldsymbol{S} \in \mathbb{R}^{p\times p}$ is the sample covariance matrix, $k = 1, \cdots, p$. When t is small enough, some elements of $\boldsymbol{\xi}_k$ will be zero.

Zou, Hastie, and Tibshirani proposed a sparse principal component algorithm from the perspective of the *closest linear manifold approximation*. Recall that if $\boldsymbol{V}_k = (\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k) \in \mathbb{R}^{p\times k}$ has orthonormal columns, then the orthogonal projection onto the column space $C(\boldsymbol{V}_k) = \operatorname{span}(\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k)$ of $\boldsymbol{V}_k$ is exactly $\operatorname{proj}_{C(\boldsymbol{V}_k)} = \sum_{j=1}^k \boldsymbol{v}_j \boldsymbol{v}_j' = \boldsymbol{V}_k \boldsymbol{V}_k'$. Consider centered observations $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \in \mathbb{R}^p$, or set $\boldsymbol{x}_i \leftarrow \boldsymbol{x}_i - \overline{\boldsymbol{x}}$. The sparse PCA criterion defined as

$$\min_{\boldsymbol{A},\boldsymbol{B}\in\mathbb{R}^{p\times k}} \left\{ \sum_{i=1}^n \|\boldsymbol{x}_i - \boldsymbol{A}\boldsymbol{B}'\boldsymbol{x}_i\|_2^2 + \lambda \sum_{j=1}^k \|\boldsymbol{b}_j\|_2^2 + \sum_{j=1}^k \lambda_{1,j} \|\boldsymbol{b}_j\|_1 \right\} \quad \text{s.t.} \quad \boldsymbol{A}'\boldsymbol{A} = \boldsymbol{I}_k$$

allows for an efficient computation of sparse loadings, where $\boldsymbol{b}_j$'s are the columns of $\boldsymbol{B}$, and λ and $\lambda_{1,j}$'s are non-negative tuning parameters. The normalized columns of the optimal $\boldsymbol{B}$ are taken as the loadings of the first k sparse principal components.

3.2 High-Dimensional Factor Model

High-dimensional measurements are frequently *dependent*, since these variables often measure similar things, such as aspects of economics or personal health. To model the dependence, factor models are frequently employed. Note that factor analysis, or more generally *latent structure learning*, can also be regarded as unsupervised learning.

Suppose that $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_n$ are p-dimensional i.i.d. observations drawn from the m-**factor model**

$$\underset{(p\times 1)}{\boldsymbol{X}} = \underset{(p\times 1)}{\mathbb{E}\boldsymbol{X}} + \underset{(p\times m)}{\boldsymbol{L}}\underset{(m\times 1)}{\boldsymbol{F}} + \underset{(p\times 1)}{\boldsymbol{\varepsilon}},$$

where $\boldsymbol{L}$ is a constant matrix consisting of *factor loadings*, $\boldsymbol{F}$ is a random vector of *common factors*, and $\boldsymbol{\varepsilon}$ is a random vector of errors. For identifiablity, assume that $\mathbb{E}\boldsymbol{F} = \boldsymbol{0}_m$, $\operatorname{Var}(\boldsymbol{F}) = \boldsymbol{I}_m$, and $\boldsymbol{L}'\boldsymbol{L}$ is diagonal. Assume further that $\operatorname{Cov}(\boldsymbol{F}, \boldsymbol{\varepsilon}) = \boldsymbol{0}_{m\times p}$. It follows the *covariance structure*

$$\boldsymbol{\Sigma} = \boldsymbol{L}\boldsymbol{L}' + \boldsymbol{\Psi}, \quad \text{where} \quad \boldsymbol{\Sigma} = \operatorname{Var}(\boldsymbol{X})\ \&\ \boldsymbol{\Psi} = \operatorname{Var}(\boldsymbol{\varepsilon}).$$

This exhibits a *low-rank plus sparse* structure, and leads naturally to the following *penalized least-squares* problem (a.k.a. **robust PCA**):

$$\min_{\boldsymbol{\Theta},\boldsymbol{\Psi}} \left\{ \|\boldsymbol{S} - \boldsymbol{\Theta} - \boldsymbol{\Psi}\|_{\mathrm{F}}^2 + \lambda \|\boldsymbol{\Theta}\|_* + \lambda\nu \sum\nolimits_{j\neq k} |\psi_{jk}| \right\},$$

where $\boldsymbol{S}$ is the sample covariance matrix, $\|\bigstar\|_{\mathrm{F}} = \sqrt{\operatorname{tr}(\bigstar'\bigstar)}$ is the Frobenius norm, $\|\bigstar\|_* = \sum |\lambda_j(\bigstar)|$ is the nuclear norm, ψ_{jk} is the (j,k) element of Ψ, and $\lambda, \nu \ge 0$ are tuning parameters. The penalties encourage the low-rankness of $\boldsymbol{\Theta} = \boldsymbol{L}\boldsymbol{L}'$ and the sparseness of $\boldsymbol{\Psi}$.

To appreciate the low-rankness of $\boldsymbol{L}$, see Exercise 8 for example.

A Exercises and Term Project

A.1 Exercises

1. (SVM and quadratic optimization problem). Given data $(\boldsymbol{X}_i, Y_i) \in \mathbb{R}^p \times \{\pm 1\}$, $i = 1, \cdots, n$, show that $\{\boldsymbol{x} \in \mathbb{R}^p : \hat{\alpha} + \boldsymbol{x}'\hat{\boldsymbol{\beta}} = 0\} = \{\boldsymbol{x} \in \mathbb{R}^p : \tilde{\alpha} + \boldsymbol{x}'\tilde{\boldsymbol{\beta}} = 0\}$, where

$$(\hat{\alpha}, \hat{\boldsymbol{\beta}}) \in \underset{(\alpha,\boldsymbol{\beta})\in\mathbb{R}\times\mathbb{S}^{p-1}}{\arg\max}\ M \text{ s.t.} \begin{cases} (\alpha + \boldsymbol{X}_i'\boldsymbol{\beta})Y_i \geq (1-\xi_i)M, \\ \xi_i \geq 0, \quad \sum \xi_i \leq B, \end{cases}$$

and

$$(\tilde{\alpha}, \tilde{\boldsymbol{\beta}}) \in \underset{(\alpha,\boldsymbol{\beta})\in\mathbb{R}\times\mathbb{R}^p}{\arg\min} \|\boldsymbol{\beta}\|_2^2 \text{ s.t. } \sum [1 - (\alpha + \boldsymbol{X}_i'\boldsymbol{\beta})Y_i]_+ \leq B,$$

where $t_+ = \max\{t, 0\}$ denotes the positive part of $t \in \mathbb{R}$.

2. (Fisher consistent loss functions). Let $\boldsymbol{X}$ be a random p-vector and Y be a random indicator taking values in $\{\pm 1\}$. Recall that the hinge loss function is defined as $L^{\text{hinge}}(\hat{y}, y) = (1 - \hat{y}y)_+$. First, show the Bayes rule

$$\arg\min_{f(\boldsymbol{x})} \mathbb{E}[L^{\text{hinge}}(f(\boldsymbol{X}), Y)|\boldsymbol{X} = \boldsymbol{x}] = \text{sign}(p_+(\boldsymbol{x}) - p_-(\boldsymbol{x})),$$

where $p_\pm(\boldsymbol{x}) = \mathbb{P}(Y = \pm 1|\boldsymbol{X} = \boldsymbol{x})$. Accordingly, explain your intuition about SVMs in no more than two sentences. A loss function ϕ is said to be **Fisher consistent** if $f_\phi^*(\boldsymbol{x}) = \arg\min_{f(\boldsymbol{x})} \mathbb{E}[\phi(f(\boldsymbol{X})Y)|\boldsymbol{X} = \boldsymbol{x}]$ satisfies that

$$\text{sign}(f_\phi^*(\boldsymbol{x})) = \text{sign}(p_+(\boldsymbol{x}) - p_-(\boldsymbol{x})), \quad \forall \boldsymbol{x} \in \mathbb{R}^p.$$

Verify that $\phi^{\text{exp}}(t) = \mathrm{e}^{-t}$ and $\phi^{\text{logit}}(t) = \log(1 + \mathrm{e}^{-t})$ are both Fisher consistent by determining f_ϕ^*. Finally, prove that a convex function ϕ is Fisher consistent if it is differentiable with $\phi'(0) < 0$.

3. (Impurity measures). Consider the Gini index, misclassification error, and cross-entropy in a simple classification setting with two classes labeled by 0 and 1. Create a single plot that displays each of these quantities as a function of $\hat{p}_{m1} = 1 - \hat{p}_{m0}$, possibly with suitable scaling.

4. (Variance reduction by averaging). Let ξ_b, $b = 1, \cdots, B$, be identically distributed random variables with mean μ and variance σ^2. Suppose that the correlation coefficients between distinct ξ_b's are all $\rho \in (0, 1)$. Show that the sample mean $\bar{\xi}_B = \frac{1}{B}\sum_{b=1}^B \xi_b$ satisfies $\mathbb{E}\,\bar{\xi}_B = \mu$ and

$$\text{Var}\left(\bar{\xi}_B\right) = \rho\sigma^2 + \frac{1-\rho}{B}\sigma^2.$$

5. (Within-group sum of squares). Let $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \in \mathbb{R}^p$, and C be a function assigning points to clusters $1, \cdots, K$. Show that for each $k = 1, \cdots, K$,

$$\frac{1}{n_k(C)} \sum_{i,j:C(\boldsymbol{x}_i)=C(\boldsymbol{x}_j)=k} \|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2^2 = 2 \sum_{i:C(\boldsymbol{x}_i)=k} \|\boldsymbol{x}_i - \hat{\boldsymbol{\mu}}_k\|_2^2,$$

where $n_k(C) = \#\{i : C(\boldsymbol{x}_i) = k\}$ and $\hat{\boldsymbol{\mu}}_k = \frac{1}{n_k(C)}\sum_{i:C(\boldsymbol{x}_i)=k} \boldsymbol{x}_i$. Also, determine

$$\min_{\boldsymbol{\mu}_k \in \mathbb{R}^p} \sum_{i:C(\boldsymbol{x}_i)=k} \|\boldsymbol{x}_i - \boldsymbol{\mu}_k\|_2^2.$$

6. (K-means clustering). Write your own K-means function in R. It should have the following form:

```
my.kmeans = function(x, centers, max.iter=20) { ... }
```

where it takes the arguments:

- `x`, an $n \times p$ matrix with the n observations along its rows;
- `centers`, a $K \times p$ matrix giving the K initial guesses for the cluster centers;
- `max.iter`, a number (defaults to 20) giving the maximum number of iterations before quitting.

It should return:

- `centers`, a $K \times p$ matrix containing the K final cluster centers;
- `cluster`, a vector of length n indicating the cluster assignment for each point;
- `num.iter`, the number of iterations taken by the algorithm.

As a check, compare your algorithm `my.kmeans` to the standard `kmeans` function in R, by passing both functions the same data and the same initial centers. (Remember to use `algorithm="Lloyd"` in `kmeans`.)

7. (Marčenko-Pastur distribution). Verify the Marčenko-Pastur theorem numerically. Then show that

$$\int_{\gamma_-}^{\gamma_+} \frac{t}{\lambda - t} \frac{\sqrt{(\gamma_+ - t)(t - \gamma_-)}}{2\pi\gamma t} \, \mathrm{d}t = \frac{1}{4\gamma} \left[2\lambda - (\gamma_- + \gamma_+) - 2\sqrt{(\lambda - \gamma_-)(\lambda - \gamma_+)} \right],$$

where $\gamma \in (0, 1)$, $\gamma_\pm = (1 \pm \sqrt{\gamma})^2$ and $\lambda > \gamma_+$. See also djalil.chafai.net/blog/2011/01/29. Denote by $\frac{1}{\beta}$ the above integral. Express λ as a function of β and γ.

8. (Low-rank factor model). Consider the 1-factor model

$$\underset{(p\times 1)}{\boldsymbol{X}_i} = \underset{(p\times 1)}{\boldsymbol{\ell}} \underset{(1\times 1)}{f_i} + \underset{(p\times 1)}{\boldsymbol{\varepsilon}_i}, \quad \boldsymbol{\varepsilon}_i \sim \mathcal{N}_p(\mathbf{0}_p, \boldsymbol{I}_p),$$

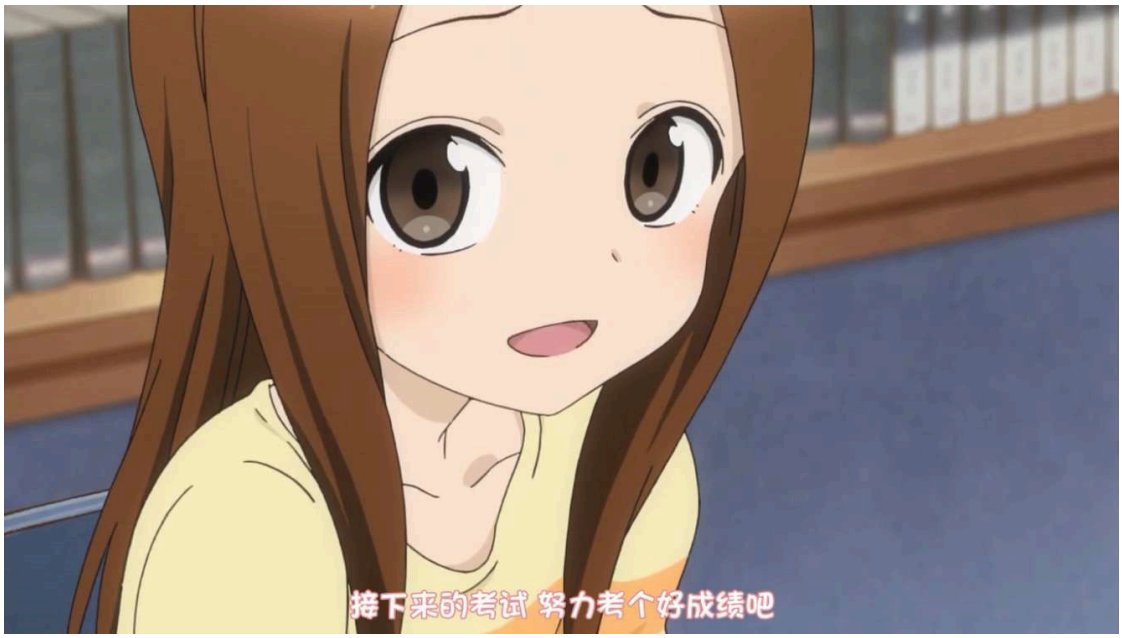
where the observations $\boldsymbol{X}_i$, $i = 1, 2, \cdots$, may not be independent. Assume that $\boldsymbol{\ell} \neq \mathbf{0}_p$ is known. Determine

$$\hat{f}_i = \arg\min_{f \in \mathbb{R}} \|\boldsymbol{X}_i - \boldsymbol{\ell} f\|_2^2 .$$

Show that

$$\mathbb{E} \max_{i \le n} |\hat{f}_i - f_i|^2 \le C \log(n) / \|\boldsymbol{\ell}\|_2^2$$

for an absolute constant $C > 0$. Particularly, C doesn't depend on p.
Hint: $\exp(\max_{i\le n} x_i) \le \sum_{i \le n} \exp(x_i)$.



A.2 Term Project

The following are guidelines for the term project in the Multivariate Statistical Analysis course.

The problem is to classify a set of 100 songs, and predict whether a specified person would like them or not, with the help from a training dataset of 650 songs. You are expected to try some (or all) classification methods from the course and evaluate their performance on the problem.

1. The dataset is available as `song_data.csv` at https://github.com/kellty/multiSTAT. The columns in the table represent extracted features, as described below. The first 650 rows serve as the training data, and the last 100 rows serve as the test data.

   - **acousticness**: A confidence measure from 0.0 to 1.0 of whether the track is acoustic. 1.0 represents high confidence the track is acoustic.
   - **danceability**: Danceability describes how suitable a track is for dancing based on a combination of musical elements including tempo, rhythm stability, beat strength, and overall regularity. A value of 0.0 is least danceable and 1.0 is most danceable.
   - **duration**: The duration of the track in milliseconds.
   - **energy**: Energy is a measure from 0.0 to 1.0 and represents a perceptual measure of intensity and activity. Typically, energetic tracks feel fast, loud, and noisy. For example, death metal has high energy, while a Bach prelude scores low on the scale. Perceptual features contributing to this attribute include dynamic range, perceived loudness, timbre, onset rate, and general entropy.
   - **instrumentalness**: Predicts whether a track contains no vocals. "Ooh" and "aah" sounds are treated as instrumental in this context. Rap or spoken word tracks are clearly "vocal". The closer the instrumentalness value is to 1.0, the greater likelihood the track contains no vocal content. Values above 0.5 are intended to represent instrumental tracks, but confidence is higher as the value approaches 1.0.
   - **key**: The key the track is in. Integers map to pitches using standard Pitch Class notation. E.g. 0 = C, 1 = C major/D minor, and so on.
   - **liveness**: Detects the presence of an audience in the recording. Higher liveness values represent an increased probability that the track was performed live. A value above 0.8 provides strong likelihood that the track is live.
   - **loudness**: The overall loudness of a track in decibels (dB). Loudness values are averaged across the entire track and are useful for comparing relative loudness of tracks. Loudness is the quality of a sound that is the primary psychological correlate of physical strength (amplitude). Values typical range between -60 and 0 db.
   - **mode**: Mode indicates the modality (major or minor) of a track, the type of scale from which its melodic content is derived.
   - **speechiness**: Speechiness detects the presence of spoken words in a track. The more exclusively speech-like the recording (e.g. talk show, audio book, poetry), the closer to 1.0 the attribute value. Values above 0.66 describe tracks that are probably made entirely of spoken words. Values between 0.33 and 0.66 describe tracks that may contain both music and speech, either in sections or layered, including such cases as rap music. Values below 0.33 most likely represent music and other non-speech-like tracks.
   - **tempo**: The overall estimated tempo of a track in beats per minute (BPM). In musical terminology, tempo is the speed or pace of a given piece and derives directly from the average beat duration.
   - **time_signature**: An estimated overall time signature of a track. The time signature (meter) is a notational convention to specify how many beats are in each bar (or measure).
   - **valence**: A measure from 0.0 to 1.0 describing the musical positiveness conveyed by a track. Tracks with high valence sound more positive (e.g. happy, cheerful, euphoric), while tracks with low valence sound more negative (e.g. sad, depressed, angry).
   - **label**: 1=LIKE and 0=DISLIKE.

2. For each method you decide to explore, you should do the following:

   (a) Implement the method. We suggest R, and you may write your own code or use packages.

   (b) Tune the method to perform well.

   (c) Evaluate its performance using, e.g., cross validation with 0-1 loss.

   Once you have completed the aforementioned tasks, you should with a good motivation select which method you decide to use "in production". Run it on the test data (for which you pretend not to know the true labels) and record the error rate, which will be an important factor in grading. Please note that a perfect classifier tends to be cheating.

3. You should summarize your work by writing a report, which will be reviewed by your teaching assistant. The report should include the following:

   (1) A brief introduction to the problem.

   (2) A concise description of each of the considered methods, and how they are applied to the problem. Please use your own words. Writing concise summaries are for your own learning, and their quality is crucial to obtain a good grade.

   (3) How the methods were applied to the data (which inputs were used, if the inputs were considered as qualitative or quantitative, how parameters were tuned, etc), including motivations of the choices.

   (4) Your evaluation of how well each method performs on the problem.

   (5) Which method you decided to use "in production", and your (good) argument for your choice.

   (6) How well your method performs "in production".

   (7) Your conclusions.

   (8) Appropriate references if any.

   (9) All code needed to reproduce your reported findings (in an appendix).

4. Your report should be submitted as a PDF-file of no longer than 6 pages (not counting the reference list and code appendix), together with the LaTeX source file.