A Novel Statistical Procedure Towards the Discovery of the Higgs Boson

by

Fadi Atieh

B.S. Mathematics and Computer Science and Engineering
Massachusetts Institute of Technology, 2020

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2022

Author .....
Department of Electrical Engineering and Computer Science
December 9, 2021

Certified by .....
Yury Polyanskiy
Associate Professor
Thesis Supervisor

Accepted by .....
Katrina LaCurts
Chair, Master of Engineering Thesis Committee

A Novel Statistical Procedure Towards the Discovery of the Higgs Boson

by

Fadi Atieh

Submitted to the Department of Electrical Engineering and Computer Science
on December 9, 2021, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

For years, physicists hypothesized the existence of the Higgs Boson; a fundamental particle in the standard model of physics playing a crucial role in the understanding of the electroweak force. However, it took almost 50 years of technological advancements until its discovery was empirically announced in 2012 [6]. The discovery was statistical in nature and relied on analyzing huge amounts of data provided by the LHC (Large Hadron Collider) at CERN. In this thesis, we propose a novel hypothesis testing approach leading to the rejection of the null hypothesis that the Higgs Boson doesn't exist. We use real data recorded at the LHC, provide theoretical build, and back it with implementation/experimentaion. Finally, we contrast our approach with the one used in the Higgs ML Challenge [3].

Thesis Supervisor: Yury Polyanskiy
Title: Associate Professor

Acknowledgments

I have received great support throughout the entire process of building my thesis.

I would first like to thank my research supervisor professor Yury Polyanskiy. His mentorship was crucial in bringing this thesis from its initial infant stages to full completion. The discovery of the Higgs Boson is an extremely complicated procedure, and without an expert like professor Yury, one could get easily lost. Beyond this thesis, absorbing minute fractions of his technical ability and expertise throughout the discussions we had significantly upped my technical competence in statistics and mathematical analysis in general. Finally, his emphasis on independence and self-reliance proved, and will continue to prove, instrumental to my confidence in facing difficult challenges, professionally and personally, going forward.

I would also like to thank my mother whose moral support was invaluable to my mental stability and motivation. Leading up to this thesis, my journey from the war-torn Syria to MIT wouldn't have been possible without my mother's encouragement. Early on in high school, having no confidence to compete on the school level, she pushed me to participate in the Mathematics Olympiad. Unexpected was that I would reach the national level and participate internationally later on. In retrospect, this came as an almost-necessary requirement for an international student to get admitted at MIT.

Finally, I'd like to thank my uncle, who proved a host, a friend, and a father when all three were so much needed in the midst of the COVID isolation, which dominated my experience throughout my master's.

Contents

List of Figures

Chapter 1

Introduction

For a long time, the Standard Model of Physics had a missing component; the Higgs Boson. The existence of the Higgs Boson was theoretically predicted as a crucial puzzle piece in understanding the electroweak force and its associated force-carrying particles. It took half a century, nonetheless, from the theoretical establishment of the Higgs Boson until its empirical discovery was announced on the 4^{th} of July, 2012 by the ATLAS and CMS collaborations [6].

The Higgs Boson discovery was unique and non-traditional. In particular, the process didn't merely rely on expertise in theoretical physics. In fact, it was, to a large extent, a statistical procedure. This is by the nature of the experiments leading to its discovery. The experiments were conducted at the LHC (Large Hadron Collider) at CERN, where protons were collided at incredible speeds. The collision would generate sub-atomic particles which would then decay to further particles and radiation. The final products of this decay, including the products of the Higgs Boson decay, were then recorded.

Because of the highly unstable nature of the Higgs Boson, it was never measured directly; only its decay products. This left the statistical analysis of such products as the only possible way to conclude its existence. In particular, the statistical methodology, in its essence, is nothing but a hypothesis test.

In this thesis, we begin with an elaboration on the significance of the Higgs Boson as a crucial component in the Standard Model of Physics (chapter 2). We then introduce the statistical procedure used in the Higgs Boson Machine Learning challenge [3], which is a famous collaboration intended to enhance the significance level of the Higgs Boson discovery (chapter 3). Then, we introduce a novel non-parametric statistical approach, contrast it with the approach in the previous chapter, and highlight why ours might be preferable (chapter 4). The approach is backed by theory and is also implemented successfully using real data from the LHC in 2012. Finally, we'll end with a summary and a list of open research questions that remain to be answered (chapter 5).

Chapter 2

Role of the Higgs Boson in the Standard Model of Physics

The Standard Model of Physics describes matter and the forces of matter interactions through fundamental particles. There are two types of fundamental particles: matter particles and force particles. Matter particles constitute matter and they can either be leptons or quarks. On the other hand, force particles carry forces between matter particles explaining their interactions.

There are four types of fundamental forces through which matter interacts: the electromagnetic force, the strong force, the weak force, and gravity. For the first three of these forces, the force particles have been experimentally verified. Namely, photons for the electromagnetic force, gluons for the strong force, and W and Z bosons for the weak force. Gravitons have been hypothesized for gravity but haven't been discovered yet [8]. The standard model of elementry particles is shown in figure 2-1

Despite the discovery of the W, Z bosons in 1983 [2], the mechanism through which these bosons obtained mass was still rather mysterious. In particular, translating the methods of Quantum Electro-Dyanamics used to explain electromagnetic interactions at that time to weak-force interactions produced massless W, Z boson which contradicted what physcists knew through experimentation. One proposed theoretical

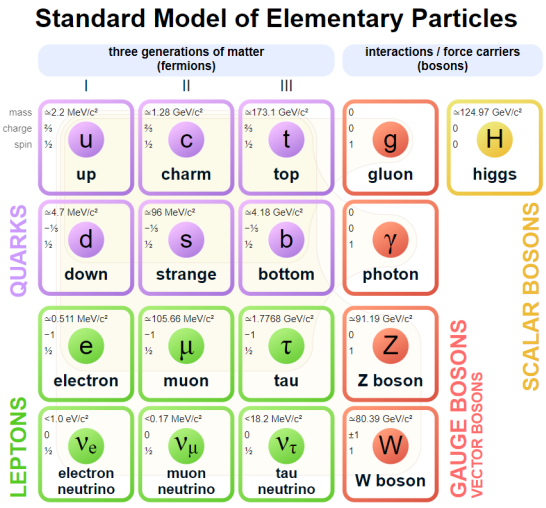

Figure 2-1: Standard Model of Elementry Particles

fix came through the Englert-Brout-Higgs-Guralnik-Hagen-Kibble mechanism, which predicted the existence of the Higgs Boson [7].

Even though the Higgs Mechanism was conceived in the 1960s, experimentally verifying the existence of the Higgs Boson required huge amounts of energy, sophisticated technology, and analysis tools that only came 50 years later. It was until 2012 when the discovery of the Higgs Boson was officially announced thanks to the CMS and ATLAS experiments done at the LHC (Large Hadron Collider) [4, 5].

2.1 Post-Discovery Experimentation

Beyond the initial discovery of the Higgs Boson, post-discovery efforts were initiated to study the various properties and decay pathways of this new particle. These efforts are crucial in increasing confidence in the initial discovery and validating the theory surrounding the Higgs Boson. For example, the Higgs ML challenge we describe in the next chapter studies a different channel than the one analyzed for the initial discovery.

Chapter 3

The Higgs Boson Machine Learning Challenge

The Higgs Boson Machine Learning Challenge was hosted at `https://www.kaggle.com/c/higgs-boson` as a competition to "explore the potential of advanced classification methods to improve the statistical significance of the [Higgs Boson discovery]"[3]. The challenge was a huge success in terms of participation with 1785 teams participating and 35772 solutions submitted. It also demonstrated the potential benefits of large scale collaborations between high-energy physicists and data scientists.

In this chapter, we go over the Higgs ML challenge looking, in some detail, into the data used, the winning models, and - most importantly - the statistical testing methodology. We end this chapter with a critique highlighting potential flaws within the methodology. This serves as a motivation for the alternative statistical testing procedure we propose in the next chapter.

3.1 Higgs Decay Channel

The Higgs Boson can decay in many different ways. Every path of decay is called a *channel* by physicists. Broadly speaking, the Higgs Boson can either decay into

further bosons or fermions. Contrary to the channels studied in the initial discovery of the Higgs Boson and the channel we study in chapter 4, the Higgs ML challenge studies the decay of the Higgs Boson into two tau leptons:

$$H \to \tau\tau$$

Recall that leptons are a subset of fermions (or matter particles). This is significant, from a physics point of view, as validating the existence of the Higgs Boson through this channel further confirms the theory surrounding it and increases confidence in the knowledge of its properties.

3.2 Data

The purpose of the challenge is to integrate machine learning techniques into the process of the Higgs Boson discovery. Data, in this case, is composed of **collision events**. Every event contains information on the final decay products of a simulated collision. Some simulated events, which are called **signal events**, come from Higgs Boson decays. While others, which are called **background events**, come from non-Higgs decays but produced final products that could have come from Higgs.

The simulated data used in the Higgs ML challenge comes from the ATLAS simulator. Access to such data became possible after it was released by the ATLAS collaboration in 2013 [4].

Simulated data is used by the participants to build machine learning models. In particular, it is split into three samples: a sample of size 250K for training, a sample of size 100K for validation, and a sample of size 450K for testing. The validation sample is used to construct a public leader board, while the testing sample is used to construct a private leader board which determines the ranking of the participants. About one third of the events in each set are signals.

3.2.1 Features

Each of the data sets has $d = 30$ features to be inputted to the classifier. The features are broadly divided into two categories: those prefixed with **PRI_** which stands for *primitive features* directly observed by the detector, and those prefixed with **DER_** which stands for *derived features* chosen by physicists to enhance the detection capability and are computed from primitive features.

Primitive features describe the momenta of particles in 3D space as well as their energy. Examples include the transverse momentum and the azimuth angle of various decay particles. Derived features, on the other hand, are mostly computed using algebraic formulae from primitive features. Examples include the invariant mass and the vector sum of momenta for various combinations of decay particles.

3.2.2 Weights

As mentioned above, about one third of each data set are signals. Of course, this is far from the frequency of signal decays relative to background decays occurring in experiments which is on the order of 10^{-5}. To account for this discrepancy, the simulated samples are weighted appropriately when training the classifiers. The weights are computed according to the following formula:

$$w_i \sim \begin{cases} p_s(x_i)/q_s(x_i), & \text{if } y_i = 1, \\ p_b(x_i)/q_b(x_i), & \text{if } y_i = 0 \end{cases}$$

where x_i is the feature vector, $y_i \in \{0, 1\}$ is the label (signal or background), $p_s(\cdot), p_b(\cdot)$ are the natural conditional distributions of feature vectors given signal/background, and $q_s(\cdot), q_b(\cdot)$ are the simulated conditional distributions of feature vectors given signal/background, respectively.

The sum of weights across each class (signal/background), and across each set is

kept fixed:

$$\sum_{i \in \mathcal{S}} w_i = N_s, \ \sum_{i \in \mathcal{B}} w_i = N_b$$

where $\mathcal{S}, \mathcal{B}$ are the index sets for signal/background, and N_s, N_b are the expected total numbers of signal/background events observed in experiments during the time of data taking.

3.3 Statistical Procedure

The purpose of the Higgs ML challenge is to enhance the significance of the Higgs Boson discovery. From a hypothesis-testing point of view, the Higgs Boson is discovered by rejecting the null hypothesis that it doesn't exist. We describe the hypothesis testing procedure on a high level in the following paragraphs.

Given a weighted training set:

$$\mathcal{D} = \{(x_1, y_1, w_1), \ldots, (x_n, y_n, w_n)\},$$

a binary classifier is trained. Assume for now that $x_i \in \mathbb{R}^d$. For a given binary classifier $g : \mathbb{R}^n \to \{0, 1\}$, a selection region $\mathcal{G}$ is defined to be:

$$\mathcal{G} = \{x : g(x) = 1\}$$

Choosing the learning the algorithm for the training is at the heart of the competition. Skipping the training step, assume we have a binary classifier g with an associated selection region $\mathcal{G}$, one obtains an unlabeled experimental sample $\mathcal{T} = (x'_1, \ldots, x'_m)$ independent of $\mathcal{D}$. One then computes the following statistic:

$$T = \sum_i 1_{x'_i \in \mathcal{G}}$$

Under the null hypothesis that the Higgs Boson doesn't exist, T is approximated to

follow a Poisson distribution Poiss(μ_b) for some μ_b. Under the alternative, however, Higgs decay adds some abundance giving distribution Poiss($\mu_b + \mu_s$) for some μ_s. Expressing $T \sim \text{Poiss}(\mu_b + \mu_s)$ where $\mu_s \in [0, \infty)$, the problem then reduces to testing the null hypothesis $\mathcal{H}_0 : \mu_s = 0$ vs. the alternative $\mathcal{H}_1 : \mu_s > 0$.

In the paper, they consider the likelihood ratio:

$$\lambda = \inf_{\mu_s} \frac{P(T|\mathcal{H}_0)}{P(T|\mathcal{H}_1)}$$

then they use Wilks's theorem to argue that:

$$q = \begin{cases} -2\ln(\lambda) & \text{if } T > \mu_b \\ 0 & \text{otherwise} \end{cases}$$

approaches a χ^2_1 distribution as $n \to \infty$. Doing some algebra, they reach the quantity:

$$Z = \sqrt{2\Big(T\ln\Big(\frac{T}{\mu_b}\Big) - T + \mu_b\Big)}$$

Which measures the significance of the test in terms of standard deviations. The goal, then, is to find a selection region $\mathcal{G}$ maximizing Z. In practice, however, given a trained classifier g, we don't know μ_b and T. So all we can hope for is an approximation. This is what the training objective AMS_c defined as:

$$\text{AMS}_c := \sqrt{2\Big((s + b + b_{\text{reg}})\ln\big(1 + \frac{s}{b + b_{reg}}\big) - s\Big)}$$

aims to do, where

$$s := \sum_{x_i : x_i \in \mathcal{G}, y_i = 1} w_i, \quad b := \sum_{x_i : x_i \in \mathcal{G}, y_i = 0} w_i$$

and $b_{\text{reg}} = 10$ is a regularization constant.

Each participant is evaluated based on their test set AMS_c classifier performance.

3.4 Critique

Although the challenge turned out to be a huge success, we discovered two potential flaws in the statistical methodology summarized as follows:

1. The test statistic used in the Higgs ML challenge is a count of the number of events in some specified region E. Note that this test is "crude" in the sense that it loses information on the likelihood of a given data point to have come from Higgs decay vs. background.

2. The discriminant function g leading to the region E above is trained using a 0/1 loss before the threshold is selected using the AMS_c objective. There is no obvious theoretical reason for using the 0/1 loss in the training.

We treat both of these in concerns in the proposed methodology described in the next chapter.

Chapter 4

Proposed Alternative Methodology

In the previous chapter, we went over the Higgs ML challenge. The challenge was intended to enhance the significance of the Higgs Boson discovery. And it was a huge success in terms of collaboration between high-energy physicists and machine learning enthusiasts.

At the end of the chapter, however, we pointed out two potential flaws within the statistical methodology used in the challenge. As an attempt to address these concerns, we propose an alternative methodology. We dedicate this chapter to go over it. On a high-level, our new methodology is different from the the one used in the Higgs ML challenge in two crucial ways:

1. Instead of counting the number of events within a specified region E as the test statistic, we use the Log-Likelihood-Ratio (LLR):

$$\mathrm{LLR}(X) := \log\Big(\frac{dQ}{dP}(X)\Big)$$

   directly, which expresses the likelihood of a given data point to have come from Higgs decay vs. background.

2. To obtain an approximation of $\mathrm{LLR}(X)$, we train a binary classifier on cross-entropy loss instead of 0/1 loss which provably converges to $\mathrm{LLR}(X)$.

4.1 Data

The data we use in our study differs from the data used in the Higgs ML challenge in a fundamental way. In particular, it describes a different decay channel. Instead of decaying into τ leptons, the Higgs Boson decays into a pair of Z bosons which further decay into 4 leptons in what is called the **Golden Channel**:

$$H \to Z\bar{Z} \to 4l$$

4.1.1 Features

The data set contains a data point for every system of 4 leptons. For each lepton, scalar features are recorded such as the kinetic energy and the momentum in 3D space. In addition to scalar features, qualitative features are recorded such as the lepton type. This resembles *primitive* features using the terminology of the previous chapter. Three additional *derived* features are used, whose impact on detection capability we measure and find significant. In total, we have $d = 35$ features.

The majority of the data is generated using a simulator. Simulated data is used in training and is labeled. In addition to the labels, appropriate weights are appended to reflect the discrepancy between the frequency of signal events within the training data set and the natural frequency of Higgs Boson decay relative to background decay. In addition to simulated data, we have real-world data recorded at the LHC during the year 2012, which we use to verify our procedure and obtain a statistically significant p-value.

4.1.2 Weights

Appending weights to the training data serves multiple purposes in our study compared to the Higgs ML challenge. In particular, in addition to the discrepancy between the natural and simulated signal-to-background frequencies, there is also a discrepancy between simulated and natural frequencies for *background components* them-

selves. That's because the background process in our dataset is actually a mixture of several sub-processes. In particular, let P be the background distribution, then:

$$P = \sum_i \alpha_i P_i$$

for some sub-components P_i and mixing ratios α_i. The compuation of the weights is detailed in the attached code.

4.2 Model Assumptions

The model is simple. Let P be the distribution of events resulting from background decay. Let Q be the distribution of events resulting from Higgs decay. Let X be the random variable describing the feature vector in $d = 35$ dimensions. We can write,

$$X \sim \pi Q + (1 - \pi)P$$

We can then formulate the following hypothesis testing problem:

$$H_0 : \pi = 0,\ H_1 : \pi \in (0, 1]$$

4.3 Choices for Statistical Tests

Suppose we have a sample of size $n = 1$, then the uniformly most powerful test for mixture distributions of P, Q relies on the log-likelihood-ratio statistic:

$$\text{LLR}(X) = \log\Big(\frac{dQ}{dP}(X)\Big)$$

by the Neyman-Pearson lemma. This suggests using $\text{LLR}(X)$ in our statistical tests. The problem is that we don't have access to LLR directly as we don't have an explicit expression for P, Q. The solution is to estimate it from the simulated data. In particular, any algorithm trained with cross-entropy loss should approximate $\text{LLR}(x)$.

More precisely, define the log-loss as usual

$$l(y,p) = y \cdot \log(\frac{1}{p}) + (1-y) \cdot \log(\frac{1}{1-p}),\ y \in \{0,1\},\ p \in [0,1].$$

Now, given $n_0 = \nu_0 n$ samples with label $y_i = 0$ and $X_i \sim P$ and $n_1 = \nu_1 n$ samples with label $y_i = 1$ and $X_i \sim Q$ we have as $n = n_0 + n_1 \to \infty$:

$$\log\Big(\frac{\hat{f}_n(X)}{1-\hat{f}_n(X)}\Big) \to \log\Big(\frac{\nu_1}{\nu_0}\Big) + \mathrm{LLR}(X)$$

Thus, training a binary classifier on log-loss can be used to obtain an approximation $\widehat{\mathrm{LLR}}(X)$.

Given an estimate $\widehat{\mathrm{LLR}}(X)$ of $\mathrm{LLR}(X)$, there are three statistical tests that come to mind: the CLT test, the Wilcoxon Rank Sum Test, and the GLRT test. We dedicate the next three sections to describe each one of them in detail.

4.3.1 CLT Test

Given $\widehat{\mathrm{LLR}}(x)$, one can estimate $a := \mathbb{E}[\widehat{\mathrm{LLR}}(X)]$, $b := \mathrm{Var}(\widehat{\mathrm{LLR}}(X))$, where $X \sim P$ from simulated data. Then, given estimates $\hat{a}, \hat{b}$, we use the Central Limit Theorem (CLT) to compute the approximate p-value:

$$p = 1 - \Phi\left(\sqrt{\frac{n}{\hat{b}}}\Big(\frac{1}{n}\sum_i \widehat{\mathrm{LLR}}(X_i) - \hat{a}\Big)\right)$$

where Φ is the CDF of $\mathcal{N}(0,1)$. Of course, for this test to be meaningful, n should be sufficiently large of for the Central Limit Theorem to kick-in. The requirement on n becomes less stringent if the distribution of $\widehat{\mathrm{LLR}}(X)$ under P is well-approximated by a Gaussian to begin with. To that end, we employ the Kolmogorov–Smirnov goodness of fit test.

4.3.2 Wilcoxon Test

Given two samples, $X_i \overset{i.i.d}{\sim} P$ from simulated data, and $\tilde{X}_j \overset{i.i.d}{\sim} \pi Q + (1-\pi)P$ from experimental data, we compute the statistic U defined:

$$U := \sum_{i,j} 1_{\widehat{\text{LLR}}(X_i) > \widehat{\text{LLR}}(\tilde{X}_j)}.$$

Under the null, $X_i, \tilde{X}_j \overset{i.i.d}{\sim} P$. Thus, each of the indicators in the previous sum follow a Bern(1/2) and the distribution approaches a Gaussian for large n, m [1]. We use this approximation in the computation of the p-value.

One important thing to note here is that, unlike the CLT test, the Wilcoxon Test is *non-parametric* in the sense that it doesn't require any assumptions on the distribution of $\widehat{\text{LLR}}(X)$ under P. What's even better, one can obtain exact $p-$values for every n, m. Although, in our study we calculate approximate p-values assuming the distribution is close to a Gaussian.

4.3.3 GLRT Test

The third test is the GLRT (Generalized Likelihood Ratio Test) T defined as follows. First define:

$$\text{obj}(\pi) := \sum_{i=1}^{n} \log(1 + \pi \cdot (e^{\widehat{\text{LLR}}(x_i)} - 1)) \quad \pi \in [0,1]$$

Then,

$$T(X^n) := \sup_{\pi \in [0,1]} \text{obj}(\pi)$$

Note as $n \to \infty$, the distribution of T approaches that of $|\mathcal{N}(0,1)|^+$ where $|\cdot|^+ = \max(0, \cdot)$.

Among all three test choices presented so far, the GLRT test requires the strongest assumptions. In particular, in addition to requiring that n is large enough to approximate the p-value using $|\mathcal{N}(0,1)|^+$, we also need $\widehat{\text{LLR}}(X)$ to be close to the true

$\mathrm{LLR}(X)$ for the test to be meaningful. We didn't verify this assumption in the study but it might be an interesting point to work on in future studies.

Technical Note: To approximate $T \sim |\mathcal{N}(0,1)|^+$, we use Wilks' theorem. This is in contrast to the χ^2_1 approximation. The reason is that the value of $\pi = 0$ under the null is at the boundary of the parameter space $[0,1]$. Note that this technical mistake was made in the Higgs ML challenge where the AMS_c was derived assuming an asymptotic χ^2_1 instead of $|\mathcal{N}(0,1)|^+$.

4.4 Results

4.4.1 Test Outcomes

From the attached code, you can see that all three statistical tests mentioned in section **4.3** are implemented. However, only one test, namely the Wilcoxon rank sum test, proves to be useful. The CLT test fails to be meaningful because we are able to reject the null hypothesis that the distribution of $\widehat{\mathrm{LLR}}(X)$ under $X \sim P$ follows a Gaussian distribution with extreme statistical significance. This removes all guarantees on the quality of the Central Limit Theorem approximation. The GLRT test, on the other hand, depends on the fact that $\widehat{\mathrm{LLR}}(X)$ is a good enough approximation of LLR. As this hasn't been verified, the GLRT test is also unreliable.

The beauty of the Wilcoxon rank sum test is that it doesn't require any assumptions on the accuracy of $\widehat{\mathrm{LLR}}(X)$. Moreover, and most importantly, it's non-parametric. So the asymptotic distribution of U is independent of the distribution of $\widehat{\mathrm{LLR}}(X)$ under P.

The p-value obtained through the Wilcoxon rank sum test is $p = 0.000318190$.

4.4.2 Detection Capability

For a given testing procedure, we define the detection capability to be the following random variable: given a resolution $\epsilon > 0$, iterate over the mixing ratios π from 0 to 1 along increments of ϵ. For every such π execute the testing procedure. Define the detection capability random variable D to be the smallest π such that the majority of tests reject the null hypothesis in the interval $[\pi - 5\epsilon, \pi + 5\epsilon]$.

Without using the derived features, we obtain a detection capability of 0.102.

4.4.3 Visual Proof for the accuracy of $\widehat{\text{LLR}}(X)$

Even though we don't verify the accuracy of $\widehat{\text{LLR}}$ quantitatively, we do have a visual proof. The visualization boils down to a GIF (Graphics Interchange Format) file. We next explain how this GIF is generated.

First, we sort the experimental samples $\{\tilde{X}_j, 1 \leq j \leq m\}$ according to their $\widehat{\text{LLR}}$ values. In other words, such that $\widehat{\text{LLR}}(\tilde{X}_{(1)}) \geq \widehat{\text{LLR}}(\tilde{X}_{(2)}) \geq \ldots \widehat{\text{LLR}}(\tilde{X}_{(n)})$. If $\widehat{\text{LLR}}$ were to be a good approximation of LLR, we would expect sorting $\tilde{X}_j$ according to their $\widehat{\text{LLR}}$ values to correspond to sorting them according to the likelihood that they have come from Higgs.

Next, for every $\tilde{X}_{(i)}$ we compute the invariant 4-mass, $\tilde{m}_{(i)}$, for the lepton system. Finally, we plot a histogram of $\tilde{m}_{(i)}$ in batches of 100 for every frame producing a GIF. On the same figure, we also plot the histogram of the invariant 4-mass of 4-lepton systems that have come from Higgs Decay as well as background decay.

What you can see clearly through the GIF is the following: at the beginning, for experimental samples with high $\widehat{\text{LLR}}(X)$ values, the histogram closely matches that of the Higgs decay. As more batches are added, or as the GIF progresses, we can see that this pattern gets *diluted*, approaching that of the background decay. This

matches what we would expect if $\widehat{\text{LLR}}(X)$ were to serve as a good approximation for $\text{LLR}(X)$.

Figure 4-3 has two plots, the one on the left (figure 4-1) is a frame early in the GIF where selected data samples have high $\widehat{\text{LLR}}(X)$ values. While the plot on the right (figure 4-2) is a frame late in the GIF where it includes almost all samples. As you can see, early in the GIF, the invariant m_{4l} histogram of experimental samples closely matches that of Higgs. Whereas late in the GIF, the pattern gets diluted.

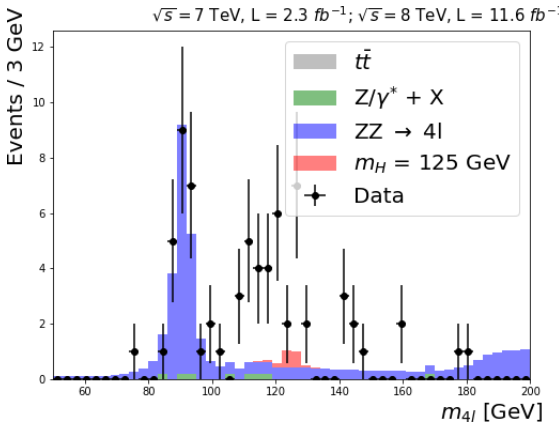

Figure 4-1: Early (High $\widehat{\text{LLR}}(X)$)

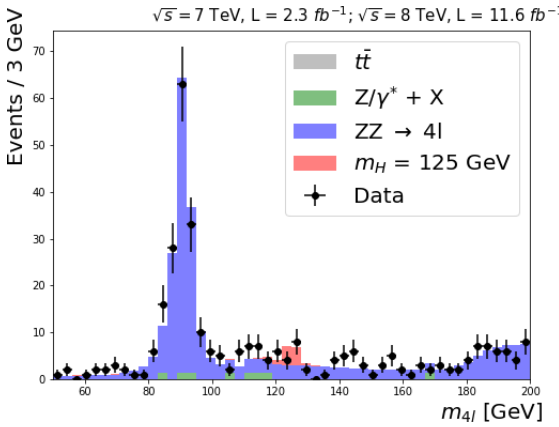

Figure 4-2: Late (Averaged $\widehat{\text{LLR}}(X)$)

Figure 4-3: No Enhanced Features Used

4.4.4 Impact of Derived Features

As we mentioned at the beginning of this chapter, the data contains three derived features engineered by physicists to enhance the statistical significance of the discovery even further.

To test how much improvement these additional features provide, we repeat the entire experiment of obtaining $\widehat{\text{LLR}}(X)$ through the training of a binary classifier and then executing the Wilcoxon rank sum test. The p-value obtained drops to $2.42350242 \cdot 10^{-6}$, an improvement of two orders of magnitude.

As for the detection capability, it enhances to 0.076

By re-generating the same GIF as we did without the enhanced features, we can see a much more pronounced effect for early frames of the GIF compared to before. (See Figure 4-6)

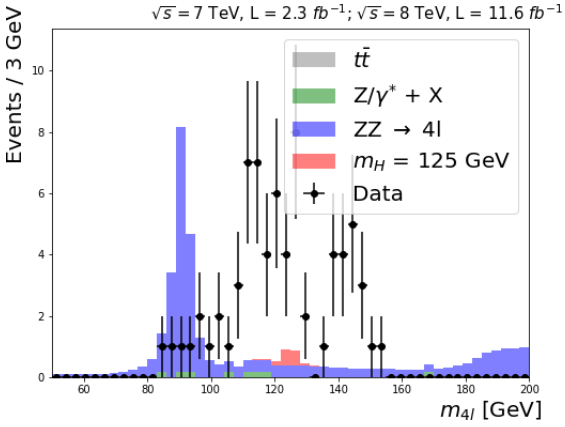

Figure 4-4: Early (High $\widehat{\text{LLR}}(X)$)

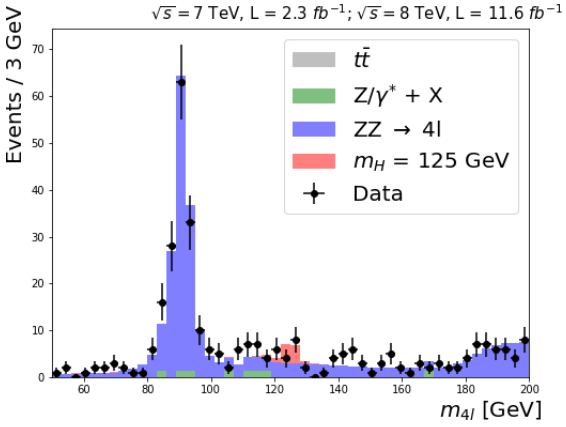

Figure 4-5: Late (Averaged $\widehat{\text{LLR}}(X)$)

Figure 4-6: Enhanced Features Added

Chapter 5

Concluding Remarks

We end this thesis by summarizing the main content and contemplating interesting future research questions.

5.1 Summary

After almost half a century of its theoretical establishment, the Higgs Boson was empirically verified in 2012 by the ATLAS and CMS collaborations [6]. Shortly after the discovery, the Higgs ML challenge [3], representing a collaboration between high-energy-physicists and ML experts, successfully produced novel statistical testing methodologies enhancing the statistical significance of the Higgs Boson discovery. Moreover, it helped increase the confidence in Higgs theory by studying a different decay channel than the one originally analyzed by physicists.

Our contribution in this thesis is the introduction of a novel ML-based approach for the discovery of the Higgs Boson. The main differences between our approach and the approach taken in the Higgs ML challenge include:

1. using the Log-Likelihood Ratio (LLR) instead of a hard-threshold in the test statistic.

2. training a neural net on log-loss to obtain an approximation of LLR instead of

the two-stage training in the Higgs ML challenge using 0/1 penality followed by AMS_c

Using the new approach, we are able to reject the null hypothesis with $p \approx 2.4 \cdot 10^{-6}$ using derived features engineered by high-energy physicists. A crucial step is estimating the Log-Likelihood Ratio (LLR) of the signal vs. background distribution through the training of a neural net. The quality of this estimation is verified visually in what boils down to a GIF (Graphics Interchange Format).

Finally, our analysis was stress-tested by introducing it as a final project for the class 6.401 (Introduction to Statistical Data Analysis) at MIT during Spring 2021. The class included 30 students. The majority of students were capable of reproducing the analysis in what was a fantastic learning experience for everyone.

5.2 Future Research Questions

While high-energy physicists use hard-thresholding, we take the opposite approach and use LLR directly. While hard-thresholding loses a lot of information, it is robust against inaccuracies in the simulator of P, Q. This can be a valid concern considering that simulating the distribution of the background/noise distributions is extremely complicated. Thus, an interesting idea is to combine both ideas in some hybrid approach. We leave this to future research.

Also, please note that throughout the analysis, we didn't impose put any assumptions on P, Q. A natural next step is constraint P, Q to belong to some - possibly non-parametric - class, and see what enhanced guarantees one might get.

Appendix A

Code Implementation

A.1 Utility Functions

Listing A.1: Util.py

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import train_test_split

def load_processes():
    # read MClist of each process and each year
    mc_higgs = pd.read_csv('../data/MC/higgs2012.csv',index_col=None, header=0)
    ## ZZ*
    mc_zz4mu = pd.read_csv('../data/MC/zzto4mu2012.csv',index_col=None, header=0)
    mc_zz2mu2e = pd.read_csv('../data/MC/zzto2mu2e2012.csv',index_col=None, header=0)
    mc_zz4e = pd.read_csv('../data/MC/zzto4e2012.csv',index_col=None, header=0)
    ## Drell-Yan
    mc_dy10 = pd.read_csv('../data/MC/dy1050_2012.csv',index_col=None, header=0)
    mc_dy50 = pd.read_csv('../data/MC/dy50_2012.csv',index_col=None, header=0)
    ## ttbar
    mc_ttbar = pd.read_csv('../data/MC/ttbar2012.csv',index_col=None, header=0)

    processes = [mc_higgs, mc_zz4mu, mc_zz2mu2e, mc_zz4e, mc_dy10, mc_dy50, mc_ttbar]
```

```
    # Add signal (signal vs. background)
    for i in range(len(processes)):
        if i==0:
            processes[i] = processes[i].assign(signal=np.ones(processes[i].shape[0]))
        else:
            processes[i] = processes[i].assign(signal=np.zeros(processes[i].shape[0]))

    return processes

def load_expr_data():
    return pd.read_csv('../data/data/clean_data_2012.csv', index_col=None, header=0)

"""
Return OH_encoder for categorial variables based on entire training data.

Input:
    processes_mc : entire mc training data
    object_cols  : categorial variables
Output:
    OH_encoder based on training data.
"""

def encoder(processes_mc, object_cols):
    reference_data = pd.concat(processes_mc, axis=0)

    OH_encoder = OneHotEncoder(handle_unknown='ignore', sparse=False)
    OH_encoder.fit(reference_data[object_cols])
    return OH_encoder


"""
Remove irrevelant predictors and One-Hot encode categorial variables.

OH_encoder      : one-Hot encoder to transform categorial variables.
object_cols     : categorial predictors
irrelevant_cols : irrelevant predictors to drop
```

```
    processes_mc     : MC data to be transformed
    expr_data        : real experimental data to transform

    Output:
        expr_data, processes_mc after being modified
    """

def trim(OH_encoder, object_cols, irrelevant_cols, processes_mc, expr_data):
    expr_data = expr_data.drop(irrelevant_cols, axis=1)

    OH_cols_data = pd.DataFrame(OH_encoder.transform(expr_data[object_cols]))

    # One-hot encoding removed index; put it back
    OH_cols_data.index = expr_data.index

    # Remove categorical columns (will replace with one-hot encoding)
    num_data = expr_data.drop(object_cols, axis=1)

    # Add one-hot encoded columns to numerical features
    expr_data = pd.concat([num_data, OH_cols_data], axis=1)

    for i in range(len(processes_mc)):
        processes_mc[i] = processes_mc[i].drop(irrelevant_cols, axis=1)

        OH_cols_mc = pd.DataFrame(OH_encoder.transform(processes_mc[i][object_cols]))

        # One-hot encoding removed index; put it back
        OH_cols_mc.index  = processes_mc[i].index

        # Remove categorical columns (will replace with one-hot encoding)
        num_mc = processes_mc[i].drop(object_cols, axis=1)

        # Add one-hot encoded columns to numerical features
        processes_mc[i] = pd.concat([num_mc, OH_cols_mc], axis=1)

    return expr_data, processes_mc
```

```
"""
Compute the weights needed for training/sampling. These weights account for the
discrepancy between the expected relative frequency of background processes and
the relative sizes of mc background processes we have at our disposal.
"""
def compute_weights():
    ## Luminosity of each year
    lumi = 11580.

    ## cross section of each process
    xsecZZ4 = 0.107
    xsecZZ2mu2e = 0.249

    xsecTTBar = 200.

    xsecDY50 = 2955.
    xsecDY10 = 10.742

    scalexsecHZZ = 0.0065

    ## Number of MC Events generated for each process
    nevtZZ4mu = 1499064
    nevtZZ4e = 1499093
    nevtZZ2mu2e = 1497445
    nevtHZZ = 299973
    nevtTTBar = 6423106
    nevtDY50 = 29426492
    nevtDY10 = 6462290

    # Compute training weights
    weights = lumi*np.array([scalexsecHZZ/nevtHZZ, xsecZZ4/nevtZZ4mu,\
                             xsecZZ2mu2e/nevtZZ2mu2e, xsecZZ4/nevtZZ4e,\
                             xsecDY10/nevtDY10, xsecDY50/nevtDY50,\
                             xsecTTBar/nevtTTBar])
```

```
    weights[0] = sum(weights[1:])

    return weights

"""
Produce a mixture sample from background processes.

Input:

processes   : background processes to sample from
frequencies : relative frequency at which to sample each background process
sample_size : size of sample to be returned

Output:
    requested sample
"""
def sample(processes, frequencies, sample_size):
    process_choice = pd.Series(np.random.choice(a=np.arange(len(frequencies)),\
                                size=sample_size, p=frequencies))
    sample_sizes = process_choice.value_counts()\
                   .reindex(np.arange(len(frequencies)), fill_value=0).sort_index()

    sample = pd.DataFrame()
    for i in range(len(processes)):
        try:
            mixing_sample = processes[i].sample(sample_sizes[i], replace=True)
        except:
            mixing_sample = pd.DataFrame()

        sample = pd.concat([sample, mixing_sample], axis=0)

    return sample
```

```
def split(processes):
    train_processes = []
    sim_processes = []
    synth_processes = []

    for i in range(len(processes)):
        train_process, sim_process = train_test_split(processes[i],\
                                                      test_size=.2, random_state=0)
        try:
            train_process, synth_process = train_test_split(train_process,\
                                                    test_size=.1, random_state=0)
        except:
            synth_process = pd.DataFrame()

        train_processes.append(train_process)
        sim_processes.append(sim_process)
        synth_processes.append(synth_process)

    return train_processes, sim_processes, synth_processes
```

A.2 Statistical Tests

Listing A.2: Data Loading and Munging

```
# Experimental data Loading (LHC 2012)
data = util.load_expr_data()

# Simulated data loading
mc_processes = util.load_processes()

# Qualtitative features
object_cols = ['PID1', 'PID2', 'PID3', 'PID4']

# Features to be discarded (note derived features 'mass', 'mass_z1', 'mass_z2'
  are initially discarded)
irrelevant_cols = ['Unnamed: 0', 'Run', 'Event', 'Q1', 'Q2', 'Q3', 'Q4', 'mass',
                   'mass_z1', 'mass_z2']

# data munging
OH_encoder = util.encoder(mc_processes, object_cols)
data, mc_processes = util.trim(OH_encoder, object_cols, irrelevant_cols, mc_processes,
                               data)

# compute weights for training
weights = util.compute_weights()

counts = np.array([process.shape[0] for process in mc_processes])

# used for generating samples from P_0
P0_frequencies = weights[1:] * counts[1:]
P0_frequencies = P0_frequencies/np.sum(P0_frequencies)

# Split simulated data to three sections:
# - training data: used in obtainint \hat{LLR(X)}
# - simulation data: used in approximating testing parameters (i.e. E[\hat{LLR}(X)]
                     under P)
# - synthetic data: used in measuring the detection capability of the tests
```

```
train_processes, sim_processes, synth_processes = util.split(mc_processes)

# Append weights to training data
for i in range(len(train_processes)):
    train_processes[i] = train_processes[i].assign(weight=weights[i]*np.ones(\
                         train_processes[i].shape[0]))


X = pd.concat(train_processes, axis=0)
training_weights = X.pop('weight')
y = X.pop('signal')
```

Listing A.3: Obtaining $\widehat{\text{LLR}}$

```
from keras.models import Sequential
from keras.layers import Dense

# Train binary classifier on cross-entropy loss to obtain \hat{LLR}(X)
def DNN_training(input_dim):
    # Construct the model
    model = Sequential()
    model.add(Dense(10, input_dim=input_dim, activation='relu'))
    model.add(Dense(10, activation='relu'))
    model.add(Dense(10, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))

    # compile the model
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

    return model


training_model = DNN_training(X.shape[1])
training_model.fit(X, y, sample_weight=training_weights, epochs=10, batch_size=64)

# Output
```

```
Fitting model
Epoch 1/10
4432/4432 [==============================] - 7s 1ms/step
- loss: 0.0129 - accuracy: 0.0980
Epoch 2/10
4432/4432 [==============================] - 6s 1ms/step
- loss: 0.0064 - accuracy: 0.0985
Epoch 3/10
4432/4432 [==============================] - 5s 1ms/step
- loss: 0.0057 - accuracy: 0.14
Epoch 4/10
4432/4432 [==============================] - 5s 1ms/step
- loss: 0.0053 - accuracy: 0.1898
Epoch 5/10
4432/4432 [==============================] - 5s 1ms/step
- loss: 0.0050 - accuracy: 0.2446
Epoch 6/10
4432/4432 [==============================] - 5s 1ms/step
- loss: 0.0051 - accuracy: 0.2339
Epoch 7/10
4432/4432 [==============================] - 6s 1ms/step
- loss: 0.0050 - accuracy: 0.2621
Epoch 8/10
4432/4432 [==============================] - 5s 1ms/step
- loss: 0.0048 - accuracy: 0.2966
Epoch 9/10
4432/4432 [==============================] - 5s 1ms/step
- loss: 0.0047 - accuracy: 0.2997
Epoch 10/10
4432/4432 [==============================] - 6s 1ms/step
- loss: 0.0049 - accuracy: 0.2763
```

```
# The prediction neural net uses linear activation in final layer instead of sigmoid
def DNN_predict(input_dim):
    model = Sequential()
```

```
    model.add(Dense(10, input_dim=input_dim, activation='relu'))
    model.add(Dense(10, activation='relu'))
    model.add(Dense(10, activation='relu'))
    model.add(Dense(1))

    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

    return model

temp_weights = [layer.get_weights() for layer in training_model.layers]

prediction_model = DNN_predict(X.shape[1])

for i in range(len(temp_weights)):
    prediction_model.layers[i].set_weights(temp_weights[i])

# Generate a simulated background sample
P0_processes = sim_processes[1:]
process_choice = pd.Series(np.random.choice(a=np.arange(len(P0_frequencies)),\
                           size=int(1e5), p=P0_frequencies))
sample_sizes = process_choice.value_counts().reindex(np.arange(len(P0_frequencies)),\
                                                     fill_value=0).sort_index()
P0_sample = pd.DataFrame()

for i in range(len(P0_processes)):
    P0_sample = pd.concat([P0_sample, P0_processes[i].sample(sample_sizes[i],\
                           replace=True)], axis=0)

h_sim = prediction_model.predict(P0_sample)
h_data = prediction_model.predict(data)


mu_hat = np.mean(h_sim)
sigma_hat = np.std(h_sim)
```

Listing A.4: CLT Test

```
from scipy.stats import norm, kstest

# Apply the Kolmogorov-Smirnov test to test whether $\hat{LLR}(X)$ could
# have come from a Gaussian under the null
gaussian_fit = norm(loc=mu_hat, scale=sigma_hat)
_, p_value = kstest(rvs=h_sim[:10000], cdf=gaussian_fit.cdf)
print("P-value for gaussian fit: {}".format(p_value))
# P-value for gaussian fit: 0.0


# We reject the null in the Komogornov-Smirnov test to we stop pursuing the CLT
# test and move on to Wilcoxon sum-rank rank sum test
```

Listing A.5: Wilcoxon Rank Sum Test

```
from scipy.stats import mannwhitneyu

_, p_value = mannwhitneyu(h_data, h_sim, alternative='greater')
print("P-value: {}".format(p_value))
# P-value: 0.00031819049796327 33
```

Listing A.6: GLRT Test

```
# GLRT Test
def obj(x):
    return -2*np.nansum(np.log(1 + x*(np.exp(h_data) - 1)))

from scipy.optimize import minimize_scalar
from scipy.stats import halfnorm

result = minimize_scalar(obj, method='bounded', bounds=(0, 1))

print("P-value: ", 1-halfnorm.cdf(-result.fun))
# P-value:  0.0
```

A.3 Adding Derived Features

Listing A.7: Integrating Derived Features

```
# Repeat entire experiment for with enhanced derived featuers

# Important for re-production
tensorflow.random.set_seed(0)
np.random.seed(0)

# re-load data here
data = util.load_expr_data()
mc_processes = util.load_processes()



# munge data here
object_cols = ['PID1', 'PID2', 'PID3', 'PID4']
irrelevant_cols = ['Unnamed: 0', 'Run', 'Event', 'Q1', 'Q2', 'Q3', 'Q4']

OH_encoder = util.encoder(mc_processes, object_cols)
data, mc_processes = util.trim(OH_encoder, object_cols, irrelevant_cols,\
                                mc_processes, data)

train_processes, sim_processes, synth_processes = util.split(mc_processes)

for i in range(len(train_processes)):
    train_processes[i] = train_processes[i].assign(weight=weights[i]*np.ones(\
                                                    train_processes[i].shape[0]))

# re-train
X = pd.concat(train_processes, axis=0)
training_weights = X.pop('weight')
y = X.pop('signal')

training_model = DNN_training(X.shape[1])

print('Fitting model')
```

```
training_model.fit(X, y, sample_weight=training_weights, epochs=10, batch_size=64)

# re-compute h_sim, h_data
P0_processes = sim_processes[1:]
process_choice = pd.Series(np.random.choice(a=np.arange(len(P0_frequencies)),
          size=int(1e5), p=P0_frequencies))
sample_sizes = process_choice.value_counts()\
               .reindex(np.arange(len(P0_frequencies)), fill_value=0).sort_index()
P0_sample = pd.DataFrame()

for i in range(len(P0_processes)):
    P0_sample = pd.concat([P0_sample, P0_processes[i]\
                .sample(sample_sizes[i], replace=True)], axis=0)

P0_sample.pop('signal')

temp_weights = [layer.get_weights() for layer in training_model.layers]

prediction_model = DNN_predict(X.shape[1])

for i in range(len(temp_weights)):
    prediction_model.layers[i].set_weights(temp_weights[i])

h_sim = prediction_model.predict(P0_sample)
h_data = prediction_model.predict(data)

# Output

 Fitting model
 Epoch 1/10
 4432/4432 [==============================] - 7s 1ms/step
 - loss: 0.0196 - accuracy: 0.1053
 Epoch 2/10
 4432/4432 [==============================] - 8s 2ms/step
 - loss: 0.0068 - accuracy: 0.1007
 Epoch 3/10
```

```
4432/4432 [==============================] - 9s 2ms/step
- loss: 0.0050 - accuracy: 0.2498
Epoch 4/10
4432/4432 [==============================] - 6s 1ms/step
- loss: 0.0041 - accuracy: 0.3945
Epoch 5/10
4432/4432 [==============================] - 6s 1ms/step
- loss: 0.0034 - accuracy: 0.4735
Epoch 6/10
4432/4432 [==============================] - 8s 2ms/step
- loss: 0.0045 - accuracy: 0.3294
Epoch 7/10
4432/4432 [==============================] - 8s 2ms/step
- loss: 0.0040 - accuracy: 0.3875
Epoch 8/10
4432/4432 [==============================] - 6s 1ms/step
- loss: 0.0029 - accuracy: 0.5059
Epoch 9/10
4432/4432 [==============================] - 6s 1ms/step
- loss: 0.0031 - accuracy: 0.4907
Epoch 10/10
4432/4432 [==============================] - 7s 1ms/step
- loss: 0.0034 - accuracy: 0.4807
```

```
# Wilcoxon Sum-Rank
_, p_value = mannwhitneyu(h_data, h_sim, alternative='greater')
print("Wilcoxon P-value: {}".format(p_value))

# GLRT
def obj(x):
    return -2*np.nansum(np.log(1 + x*(np.exp(h_data) - 1)))

result = minimize_scalar(obj, method='bounded', bounds=(0, 1))

print("GLRT P-value: {}\n\n".format(1-norm.cdf(np.sqrt(-result.fun))))
```

```
# Wilcoxon P-value: 2.4235024235388715e-06
# GLRT P-value: 0.0
```

A.4 Detection Capability

Listing A.8: P-value computation

```
from scipy.stats import norm, mannwhitneyu
def compute_p_value(data, h_sim, h_hat):
 n = len(data)
 h_data = h_hat.predict(data)

 # Wilcoxon Sum–Rank
 _, p_value = mannwhitneyu(h_data, h_sim, alternative='greater')
 return p_value
```

Listing A.9: Detection Capability Computation

```
def detection_capability(synth_processes, P0_proportions, h_hat, h_sim,\
                         p_threshold=.01, accuracy=500):
 recent = np.array([float('nan')]*10)
 synth_P0_sample = util.sample(synth_processes[1:], P0_proportions, int(2e4))

 for i in range(0, accuracy):
 mixing_ratio = i/accuracy

 noise_bkg_choice = pd.Series(np.random.choice([0, 1], size=300,\
                              p=[1-mixing_ratio, mixing_ratio]))
 sample_sizes = noise_bkg_choice.value_counts().\
                reindex(np.arange(2), fill_value=0).sort_index()

 synth_data = pd.DataFrame()
 synth_data = pd.concat([synth_data, synth_P0_sample.\
              sample(sample_sizes[0], replace=True)], axis=0)
 synth_data = pd.concat([synth_data, synth_processes[0]\
              .sample(sample_sizes[1], replace=True)], axis=0)

 synth_data.pop('signal')
 synth_data.pop('weight')
```

```
 p_value = compute_p_value(synth_data, h_sim, h_hat)
 recent = np.delete(recent, len(recent)-1)
 recent = np.insert(arr=recent, obj=0, values=p_value<p_threshold)
 if np.nanmean(recent)>.5:
 return mixing_ratio - 5/accuracy
 return
```

Listing A.10: Vanilla Detection Capability

```
print('Detection Capability: ', detection_capability(synth_processes,\
        P0_frequencies, h_hat, h_sim))

# Output
Detection Capability: 0.10200000000000001
```

Listing A.11: Detection Capability with Derived Features Added

```
with open('../experiment_assets/enhanced_h.npy', 'rb') as f:
 h_sim = np.load(f)
 _ = np.load(f)
 P0_frequencies = np.load(f)

h_hat = keras.models.load_model('../experiment_assets/enhanced_model')
synth_processes = []
for i in range(len(P0_frequencies)):
 synth_processes.append(pd.read_pickle('../experiment_assets/enhanced_synth_process_'\
                                       + str(i) + '.pkl'))

print('Detection Capability: ', detection_capability(synth_processes,\
        P0_frequencies, h_hat, h_sim))
# Output
Detection Capability: 0.076
```

Bibliography

[1] Marylise Julien Carine A. Bellera and James A. Hanely. *Journal of Statistics Education*, 2017.

[2] UA1 Collaboration CERN (Geneva, Switzerland). Archives of the ua1 collaboration, underground area 1 collaboration. *CERN Scientific Information Service*, 1978-1993.

[3] et al. Claire Adam-Bourdarios. *NIPS 2014 Workshop on High-energy Physics and Machine Learning*, 2015.

[4] ATLAS Collaboration. Observation of a new particle in the search for the standard model higgs boson with the atlas detector at the LHC. *Science Direct*, 2012.

[5] CMS Collaboration. Observation of a new boson at a mass of 125 gev with the cms experiment at the LHC. *Science Direct*, 2012.

[6] Karl Jakobs and Chris Seez. The higgs boson discovery. *Scholarpedia*, 2015.

[7] Tom W B Kibble. Englert-brout-higgs-guralnik-hagen-kibble mechanism. *Scholarpedia*, 2009.

[8] Christoph Paus. Jlab experiments. https://github.com/JLabMit/JLabExperiments, 2020.