Machine learning approaches to manufacturing and materials: Applications to semi-supervised, unbalanced and heterogeneous data problems

Rasika S. Karkare
Worcester Polytechnic Institute

Machine learning approaches to manufacturing and materials: Applications to semi-supervised, unbalanced and heterogeneous data problems

by

Rasika S. Karkare

A Thesis
Submitted to the Faculty
of the
WORCESTER POLYTECHNIC INSTITUTE
in partial fulfillment of the requirements for the
Degree of Master of Science
in
Data Science
by

August 2019

APPROVED:

Professor Randy C. Paffenroth, Major Advisor

Professor Xiangnan Kong, Thesis Reader

Abstract

The objective of this thesis is to use machine learning and deep learning techniques for the quality assurance of metal casting processes. Metal casting can be defined as a process in which liquid metal is poured into a mold of a desired shape and allowed to solidify. The process is completed after ejection of the final solidified component, also known as a casting, out of the mold. There may be undesired irregularities in the metal casting process known as casting defects. Among the defects that are found, porosity is considered to be a major defect, which is difficult to detect, until the end of the manufacturing cycle. When there are small voids, holes or pockets found within the metal, porosity defect occurs. It is important to control and alleviate porosity below certain permissible thresholds, depending on the product that is being manufactured. If the foundry process can be modeled using machine learning approaches, to predict the state of the casting prior to completion of the casting process, it would save the foundry the inspection and testing of the casting, which requires specific attention of the staff and expensive machinery for testing. Moreover, if the casting fails the quality test, then it would be rendered useless. This is one of the major issues for the foundries today. The main aim of this project, is to make predictions about the quality of metal cast components. We determine whether under certain given conditions and parameters, a cast component would pass or fail the quality test. Although this thesis focuses on porosity defects, machine learning and deep learning techniques can be used to model any other kinds of defects such as shrinkage defects, metal pouring defects or any metallurgical defects. The other important objective is to identify the most important parameters in this casting process, that are responsible for the porosity control and ultimately the quality of the cast component. The challenges faced during the data analysis while dealing with a small sized, unbalanced, heterogeneous and semi-supervised dataset, such as this one, are also covered. We compare the results obtained using different machine learning techniques in terms of F_1 score, precision and recall, among other metrics, on unseen test data post cross validation. Finally, the conclusions and scope for the future work are also discussed.

Acknowledgements

I would like to express my gratitude and appreciation for my advisor, Prof. Randy Paffenroth. He gave me an opportunity to pursue my Masters thesis under his guidance at Worcester Polytechnic Institute(WPI). Dr. Paffenroth has perpetually conveyed a spirit of scientific and conversant attitude towards research and scholarship to all of his students. I am immensely thankful to him for his continuous support of my M.S. study. I am also thankful to the collaborators on this project, Dr. Diran Apelian and Dr. Ning Sun for their thoughtful guidance and encouragement throughout this project.

Contents

List of Figures

List of Tables

Executive Summary

The main objective of this thesis is to make predictions about the quality of metal cast components. We determine whether under certain given conditions and parameters, a cast component would pass or fail the quality test. Moreover, we identify the most important parameters in this casting process, that are responsible for porosity control and ultimately the quality of the cast component. The main challenges in dealing with data sets such as this one, are that they are unbalanced, semi-supervised and heterogeneous. There are very few parts that fail the quality test, as compared to the ones that pass. The foundry makes an assumption about the quality label of some of the components, based on the underlying conditions, without manually testing each part. However, it is essential that we verify using machine learning and deep learning approaches, whether the components that the foundry assumes to be good quality are indeed good quality. The other challenge is that the foundry collects the data in stages, during different phases of the manufacturing cycle and the processing data is in silos. For example, there is the process data which includes the different parameters of the process such as densities, temperatures in different zones and then the chemistry data, that includes the elemental compositions of the cast components. It is important that the siloed data is merged, integrated and analyzed, in order for the foundries to understand it better and improve their decision making process with the ultimate aim of reducing scrap rates.

This executive summary provides an overview of the challenges that are found in such data sets and our approach in trying to overcome each of the challenges, in order to add value to the operations of a foundry.

In one of the novelties of the study, we implement a generative adversarial network (GAN), which is a deep learning approach to generate synthetic data, that is similar to the real data in terms of its distribution. This demonstrates the use of deep learning models for data augmentation in case of small and unbalanced datasets. We implemented a variation of a regular GAN in a semi-supervised setting. We do this to leverage the unlabeled data present in the dataset, along with the labeled data. The details of the GAN architecture and its working will be covered in the chapter on data augmentation.

This study provides novel contributions, by performing an in depth analysis of the above model using hyperparameter tuning and shows that GANS have a limitation and show a potential of overfitting. There is a limit to the amount of data that can be generated, where after a certain point, the generator generates the same data points. This increases the accuracy on the training data, however, it fails to generalize on the unseen test data.

The GAN was implemented to address the challenge of having a small sized and semi supervised dataset, by way of data augmentation.

Another challenge in dealing with metal casting datasets, is that they are heterogeneous and multimodal and often times have quite a few missing values. The missing values could be due to either a technical reason such as faulty sensors or human errors during the data collection phase. We address this issue, by implementing masked heterogeneous autoencoder and masked variational autoencoder models for missing data imputation, and show that the imputed data using our model matches the performance of the original data in terms of the area under a precision recall curve, even when the missing rate goes up to as high as 0.6.

We observed that the HMVAE model is better than the former and has a better reconstruction capability as compared to the other model across different missing rates. This can be seen in the figure shown below. The details of the loss functions, architecture and working of these models will be discussed in the chapter on missing data imputation.

Lastly, the dataset is also highly imbalanced. We implemented a number of resampling strategies to give higher weight to the minority or the failed class, which is the more important one in this context. We also tried different resampling ratios and classification schemes for the dataset.

The imbalanced data has minority class as only 6% of the total dataset. The details of the implementations will be covered in the chapter on balancing techniques. However, it is important to note that the size of the dataset is very small and more data is needed to validate the above models.

Chapter 1

Introduction

1.1 Motivation

Modern foundries have the capability of capturing large amounts of data during the metal casting manufacturing process. This data includes the casting process data, the molten metal preparation details data, simulation data, part geometry data and non-destructive testing data. However, the data is collected during different stages of the casting process. This data is kept in silos and their utility is often limited until they are integrated together. Unless there is a way to integrate, fuse and analyze the data, this would be a lost opportunity for the foundry. Figure 1.1 shows the various processing data in silos.

The cast component could have a number of irregularities. Among the defects that are found, porosity is considered to be a major defect, which is not easy to detect [10]. The below figure shows the defects that are found in metal casting processes. It can be seen that around 35% of the casting defects are caused by porosity. Also, about 65% of the defects are related to porosity. Many a times, the porosity defect is located internally in the casting and will not be exposed until the casting is machined, which makes it hard to identify. At this point in the cycle, the cast component has already accumulated significant amount of operation cost and if the casting is bad quality, then it would have to be rejected and the producers have to pay for the high rejection cost [10].

Porosity is primarily caused by solidification shrinkage or gas bubbles trapped within the casting. Moreover, when there are small voids, holes or pockets found within the

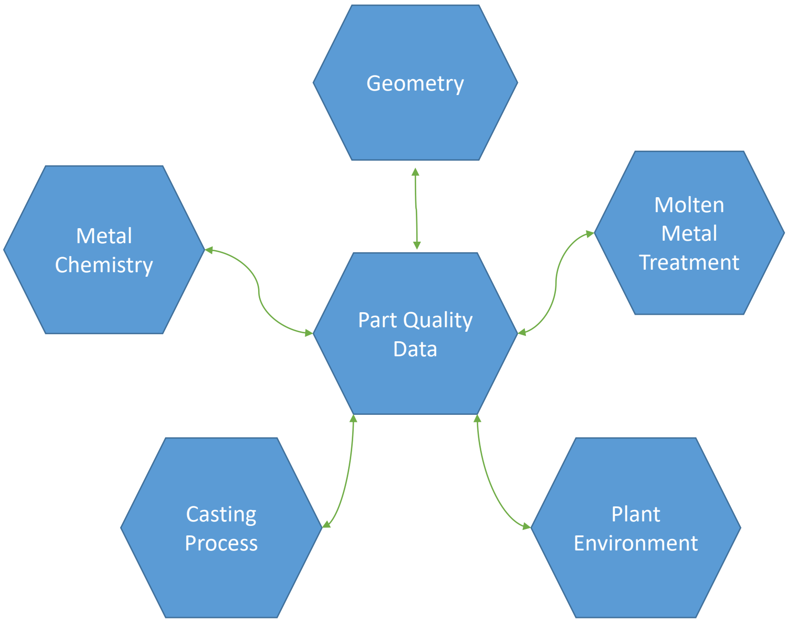


Figure 1.1: Manufacturing data is collected at different stages of the manufacturing cycle. The siloed data is shown in this figure [1].

metal, porosity defect occurs. It is important to control and alleviate porosity below certain permissible thresholds, depending on the product that is being manufactured. It is critical to find out the key factors that are responsible for porosity formation in the production. The Figure 1.3 shows an example of the porosity defect in a cast bronze bar stock [11].

The objective of this research is to overcome such challenges in the metal casting manufacturing process, so as to create a knowledge-based approach for the foundries and to improve their process cognition. Foundries as of today, have no means to analyze process data and correlate it with quality data. It is essential to assist in the overall understanding of the root cause of porosity formation and enhance this casting process at the foundries and throughout the casting supply chain.

Machine Learning based approaches have shown to deeply impact such data heavy industries. This thesis covers some of the approaches that can be exploited to build a system that could help have better control over the metal casting process and create value for the foundries. This in turn, could enhance data driven decision making for the

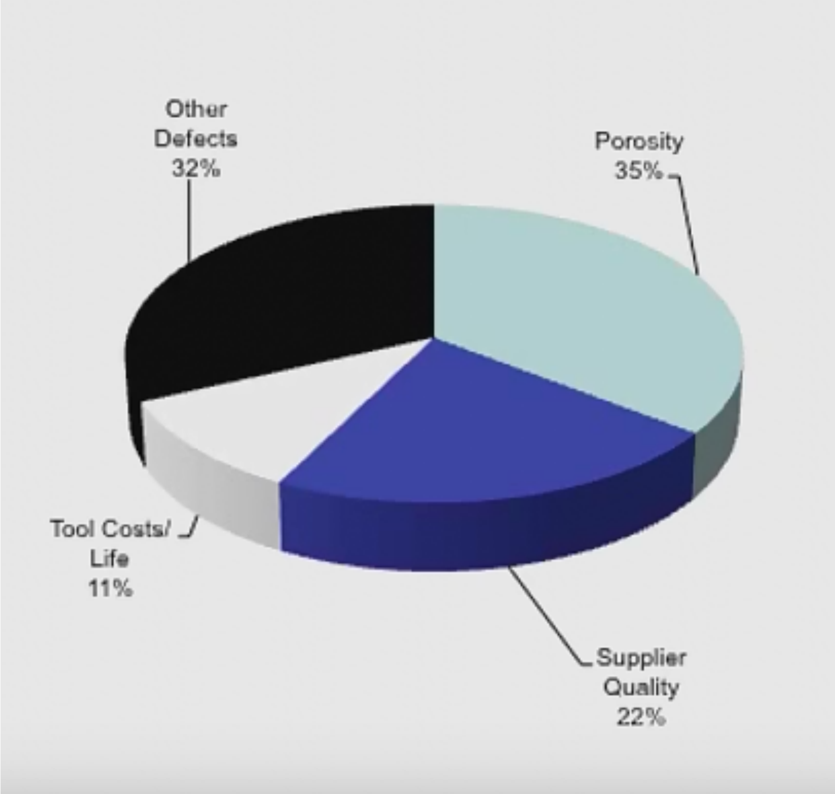


Figure 1.2: This figure shows the defects present in the metal casting process [2].

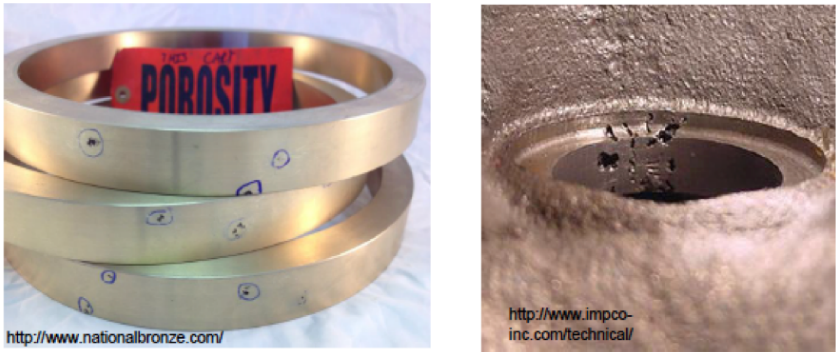
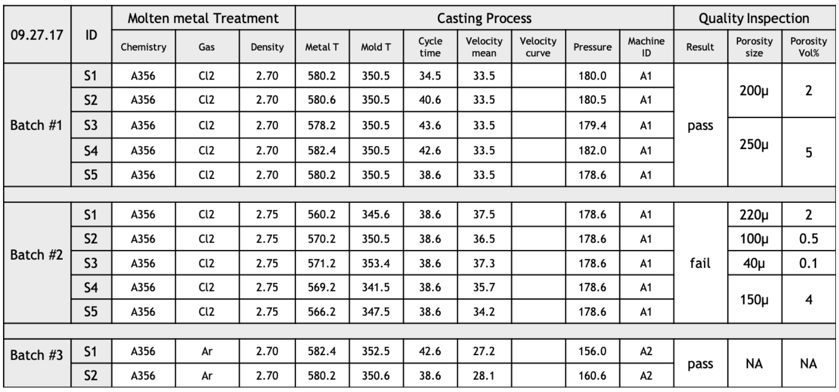

POROSITY
http://www.nationalbronze.com/
http://www.impco-inc.com/technical/

Figure 1.3: This figure shows porosity defect in a cast bronze bar stock [3].

foundries.

Furthermore, the work done in this thesis demonstrates that machine learning and deep learning approaches can provide significant improvements to the metal casting process and yields. The aim is to develop a framework that would transform the chemistry and the process related data into knowledge, which can be leveraged by the foundries.

The inputs would be raw data sets that would be pre-processed. The outputs would be predictions of the part quality along with identifying the most important parameters that are responsible for porosity levels of the cast parts and ultimately the quality of the metal cast components. The results can be verified with the domain experts who have knowledge of the entire casting process. This work relies on and promotes both the machine learning aspect as well as the human learning process. Moreover, anomalous parts can be determined based on their processing conditions. By reducing the defective parts and ultimately the scrap rate, it would eventually lead to creating smart foundries as part of the age of Industry 4.0. The figure below shows an example of the metal casting process dataset.



09.27.17	ID	Molten metal Treatment			Casting Process							Quality Inspection		
		Chemistry	Gas	Density	Metal T	Mold T	Cycle time	Velocity mean	Velocity curve	Pressure	Machine ID	Result	Porosity size	Porosity Vol%
Batch #1	S1	A356	Cl2	2.70	580.2	350.5	34.5	33.5		180.0	A1	pass	200μ	2
	S2	A356	Cl2	2.70	580.6	350.5	40.6	33.5		180.5	A1			
	S3	A356	Cl2	2.70	578.2	350.5	43.6	33.5		179.4	A1		250μ	5
	S4	A356	Cl2	2.70	582.4	350.5	42.6	33.5		182.0	A1			
	S5	A356	Cl2	2.70	580.2	350.5	38.6	33.5		178.6	A1			
Batch #2	S1	A356	Cl2	2.75	560.2	345.6	38.6	37.5		178.6	A1	fail	220μ	2
	S2	A356	Cl2	2.75	570.2	350.5	38.6	36.5		178.6	A1		100μ	0.5
	S3	A356	Cl2	2.75	571.2	353.4	38.6	37.3		178.6	A1		40μ	0.1
	S4	A356	Cl2	2.75	569.2	341.5	38.6	35.7		178.6	A1		150μ	4
	S5	A356	Cl2	2.75	566.2	347.5	38.6	34.2		178.6	A1			
Batch #3	S1	A356	Ar	2.70	582.4	352.5	42.6	27.2		156.0	A2	pass	NA	NA
	S2	A356	Ar	2.70	580.2	350.6	38.6	28.1		160.6	A2			

Figure 1.4: This figure shows an example of a metal casting dataset.

1.2 Challenges

The main challenges in trying to convert the metal casting data into knowledge are mainly as follows:

- Unbalanced dataset:

  The part quality data is extremely unbalanced. Handling unbalanced data for machine learning is a growing research area and numerous studies have been carried out to handle the skewed data set issue, since a large number of real-world datasets are highly imbalanced. Successful foundries have a large-scale production of defect free components. Owing to this, the defect free or good quality parts produced are significantly higher than the failed parts. Furthermore, it is very expensive to perform the quality inspection of all products. As a result of this, it is easy to measure most of the processing data of each casting, however to generate the quality data of the casting can be quite challenging. The below plot shows an example of the imbalance ratio of the classes.

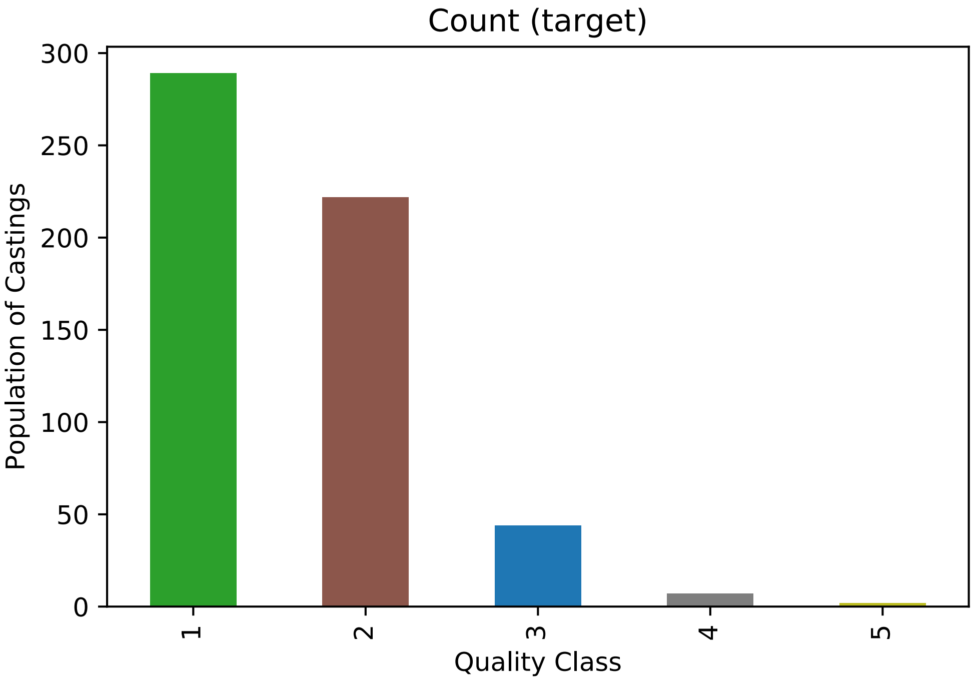
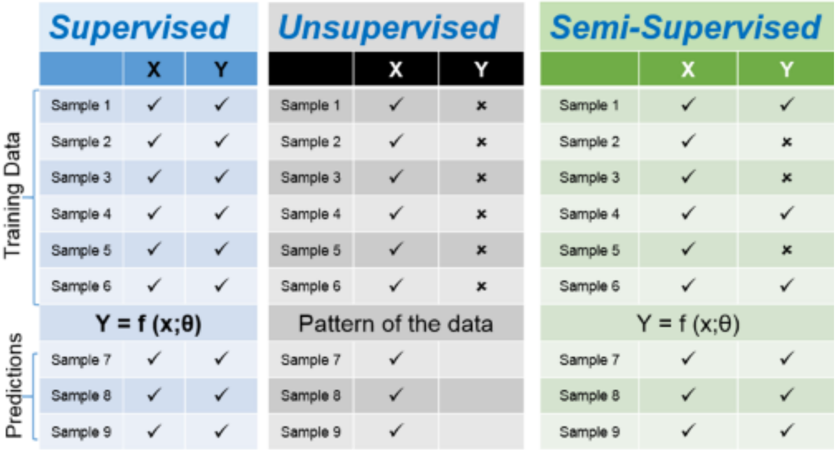


Figure 1.5: This figure shows the population of casting in each quality class. It can be seen that the dataset is highly imbalanced.

This setting makes the metal casting data fall into the category of an unbalanced and semi-supervised learning problem, which is challenging even for the state-of-the-art machine learning algorithms. The minority class features are particularly underrepresented in this case which leads to a poor performance while applying traditional machine learning algorithms. By using resampling techniques, the algorithms can learn from the structure of the failed or the minority class in the original data set and then generate new data points from this class such that the population of the failed parts will be balanced with the pass class. This way, the population of all classes can be equal to each other and can improve minority class detection.

The figure below shows a snapshot of a semi-supervised data set, where only some of the instances are labeled and some are unlabeled.



	Supervised			Unsupervised			Semi-Supervised		
		X	Y		X	Y		X	Y
Training Data	Sample 1	✓	✓	Sample 1	✓	×	Sample 1	✓	✓
	Sample 2	✓	✓	Sample 2	✓	×	Sample 2	✓	×
	Sample 3	✓	✓	Sample 3	✓	×	Sample 3	✓	×
	Sample 4	✓	✓	Sample 4	✓	×	Sample 4	✓	✓
	Sample 5	✓	✓	Sample 5	✓	×	Sample 5	✓	×
	Sample 6	✓	✓	Sample 6	✓	×	Sample 6	✓	✓
	Y = f (x;θ)			Pattern of the data			Y = f (x;θ)		
Predictions	Sample 7	✓	✓	Sample 7	✓		Sample 7	✓	✓
	Sample 8	✓	✓	Sample 8	✓		Sample 8	✓	✓
	Sample 9	✓	✓	Sample 9	✓		Sample 9	✓	✓

Figure 1.6: Semi-supervised dataset. We have the response variable only for a few parts, and do not know the quality of the other parts.

To deal with the imbalance, small size and semi supervised learning scenarios, two approaches were mainly used:

1. Data level approaches and Algorithm level approaches: In the data level resampling approaches, the minority or majority classes are either up-sampled or down-sampled respectively in order to create equal number of instances in both the classes, in order to balance the proportions. In algorithm level approaches, the algorithm is itself modified to bias the predictions towards the minority class by giving more weight to this class [12].
2. Deep learning is popularly being used these days for data augmentation of small and unbalanced datasets. In particular, generative adversarial networks (GAN) is a popular deep learning technique, used for synthetic data generation in this study. We implement a variation of a GAN in a semi-supervised setting, which can also leverage the fact that the data consists of unlabeled

data. The detailed implementation of these techniques and their comparison, will be covered in the further sections [13].

- Another challenge in dealing with metal casting datasets, is that they are heterogeneous and multimodal and often times have quite a few missing values. The missing values could be due to either a technical reason such as faulty sensors or human errors during the data collection phase. We address this issue, by implementing heterogeneous denoising and variational autoencoder models for missing data imputation, and show that the imputed data using our model matches the performance of the original data in terms of the area under a precision recall curve, even when the missing rate goes up to as high as 0.6. The figure below shows the various processing data in silos, which represent the multimodal, heterogeneous nature of the dataset. The details of the model will be covered further in the section on missing data imputation.

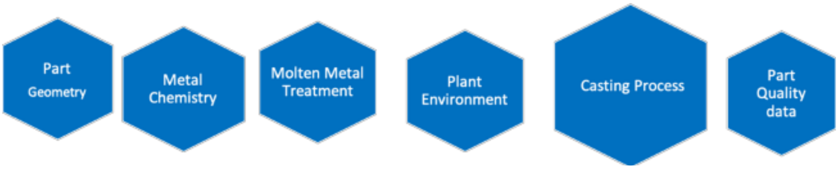


Figure 1.7: This figure shows the siloed data implying the heterogeneous and multimodal nature of the dataset [1].

1.3 Thesis Contributions

This thesis possesses the following contributions to the field of Data Science and Metal casting:

- Novel contributions to the field of metal casting by using Generative Adversarial Networks (GAN) for synthetic data generation. The generated data being notably similar to the original data belonging to both the good and the bad quality part data. The need for additional data is vital because of a very small size of such data sets.

While GANS are being popularly used nowadays for generating synthetic image data, to the best of our knowledge, this is the first study which implements a GAN in a semi-supervised setting on a manufacturing dataset such as metal casting. We have successfully been able to generate synthetic data faithful to the original data distribution and also observed that the GAN classifier gives better performance in terms of F_1score as compared to other classifiers on a balanced dataset. This setup eliminates the need for having a separate classifier for supervised learning, since the discriminator itself is used as a classifier.

- Although there have been studies earlier that have used GAN in a semi-supervised context on image datasets, we also perform an in depth analysis using hyperparameter tuning on the semi-supervised GAN model and demonstrate its potential for overfitting across multiple training epochs and changing model architectures.
- We implemented a HMAE and HMVAE using an indicator matrix to mask random fraction of values in the dataset and reconstruct those values using backpropogation. Our results validate the success of our models by comparing them to naive techniques such as mean imputation. The performance using the autoencoder scheme almost matches that of the original data, even when the missing data rate goes up to as high as 0.6. This will help us deal with datasets with a high proportion of missing values, by not discarding the samples with incomplete information, and not reducing the size of the dataset for analysis. We demonstrate the superiority of our model on both, an open source toy dataset for regression as well as the metal casting dataset for classification.
- We use neural networks, with stratified cross validation to find and save the best weights for the model that minimizes the validation loss. We weigh the classes based on the frequency of samples of each class present in the dataset. This reduces the bias towards the majority class by giving more weight to the minority class of the unbalanced dataset. We then use the saved weights to test the performance on testing data. This model shows improved performance on the testing data in terms of the recall on the minority class, as compared to any other resampling techniques used for balancing the data. We target models with a high recall on the minority class, which is the more important class in this context.

- Robust combinations of techniques to find important features responsible for predicting the quality of the metal cast component were implemented. The features found using these techniques have been verified with the foundry engineers and are in agreement with the most important features, based on their domain knowledge.

Chapter 2

Data pre-processing

2.1 Merging the raw data sets

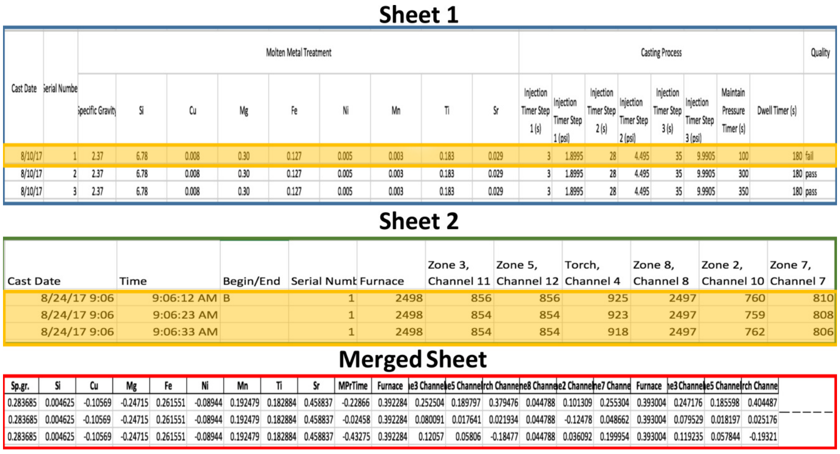
This research consists of analysis on two metal casting data sets, described as follows:

- The first data set is a marine engine block data set. As can be seen in the image below, the raw data, consisted of two sheets. The first sheet consisted of 19 columns and 176 rows. The second sheet consisted of 11849 rows and 11 columns. The two sheets needed to be merged in such a way that, every instance in the final merged sheet would contain information about one particular cast component based on the serial number and date combination from the original two sheets for that component.

- The first sheet consisted of information for the component, for the molten metal treatment and casting process along with the response variable i.e. quality pass or fail. The second sheet consisted of the cycle information for each component from sheet 1. Each cast component has a different cycle time, depending on its composition. Cycle time here refers to the total time that it takes a particular component to go from the liquid molten phase to the solidification state when the final cast part is ejected. For consistency, in terms of the dimensions of the final merged sheet, the cycle time was chosen as 450 seconds for each component with each reading for temperature being taken every 10 seconds. In the second sheet, the start of the cycle time is given by B and the end of the cycle time is given as

E.

The below image shows the snapshot of the merging process for this dataset.



Sheet 1

Cast Date	Serial Numbe	Molten Metal Treatment									Casting Process								Quality
		Specific Gravity	Si	Cu	Mg	Fe	Ni	Mn	Ti	Sr	Injection Timer Step 1 (s)	Injection Timer Step 1 (psi)	Injection Timer Step 2 (s)	Injection Timer Step 2 (psi)	Injection Timer Step 3 (s)	Injection Timer Step 3 (psi)	Maintain Pressure Timer (s)	Dwell Timer (s)	
8/10/17	1	2.37	6.78	0.008	0.30	0.127	0.005	0.003	0.183	0.029	3	1.8995	28	4.495	35	9.9905	100	180	fail
8/10/17	2	2.37	6.78	0.008	0.30	0.127	0.005	0.003	0.183	0.029	3	1.8995	28	4.495	35	9.9905	300	180	pass
8/10/17	3	2.37	6.78	0.008	0.30	0.127	0.005	0.003	0.183	0.029	3	1.8995	28	4.495	35	9.9905	350	180	pass

Sheet 2

Cast Date	Time	Begin/End	Serial Numb	Furnace	Zone 3, Channel 11	Zone 5, Channel 12	Torch, Channel 4	Zone 8, Channel 8	Zone 2, Channel 10	Zone 7, Channel 7
8/24/17 9:06	9:06:12 AM	B	1	2498	856	856	925	2497	760	810
8/24/17 9:06	9:06:23 AM		1	2498	854	854	923	2497	759	808
8/24/17 9:06	9:06:33 AM		1	2498	854	854	918	2497	762	806

Merged Sheet

Sp.gr.	Si	Cu	Mg	Fe	Ni	Mn	Ti	Sr	MPrTime	Furnace	e3 Channe	e5 Channe	rch Channe	e8 Channe	e2 Channe	e7 Channe	Furnace	e3 Channe	e5 Channe	rch Channe
0.283685	0.004625	-0.10569	-0.24715	0.261551	-0.08944	0.192479	0.182884	0.458837	-0.22866	0.392284	0.252504	0.189797	0.379476	0.044788	0.101309	0.255304	0.393004	0.247176	0.185598	0.404487
0.283685	0.004625	-0.10569	-0.24715	0.261551	-0.08944	0.192479	0.182884	0.458837	-0.02458	0.392284	0.080091	0.017641	0.021934	0.044788	-0.12478	0.048662	0.393004	0.079529	0.018197	0.025176
0.283685	0.004625	-0.10569	-0.24715	0.261551	-0.08944	0.192479	0.182884	0.458837	-0.43275	0.392284	0.12057	0.05806	-0.18477	0.044788	0.036092	0.199954	0.393004	0.119235	0.057844	-0.19321

Figure 2.1: This figure shows the merged data from the two raw sheets.

The dimensions of the final merged sheet were 452 rows and 256 columns. The label prediction in this data set being a binary classification problem. The total number of samples that were tested were 224. Out of these, 205 passed the quality test whereas, only 19 failed the test. The rest of the samples were not tested and were assumed to be good quality parts. However, since the 228 parts were based on an assumption of passing the test without actually being tested, the quality of these parts were predicted using machine learning techniques, in order to ensure that they were indeed good quality parts. Moreover, if the company is shipping out the product to their customer, under the assumption that it is a good quality part and it fails after being delivered, it would have a tremendous impact on the company not only in terms of finances, but it would also put the company reputation at stake. Also, the ratio of the 2 part qualities shows that the actual tested parts that failed, are a meager 0.09 percent of the total number of parts in the final merged data set.

The second dataset is that of a silicon wafer ion implantation system, obtained from a foundry in Palmer, MA. The raw data, once again consisted of 2 sheets with the process and the chemistry parts respectively. We merged the 2 sheets based on serial number and date combinations to obtain the final merged sheet. The final merged sheet consisted of 565 rows and 20 features. The response variable consisted of 5 part qualities. Although it is a multi-class classification scheme, we implemented a number of different schemes including binary using the most common and the least common classes as well one-vs-all schemes to validate the performance of our models. The quality is in terms of porosity levels where class 1 is the best or good quality parts in terms of porosity and class 5 is the worst and samples of this class will need to be scraped immediately. The proportion of castings in the dataset belonging to each class is shown in the figure below:

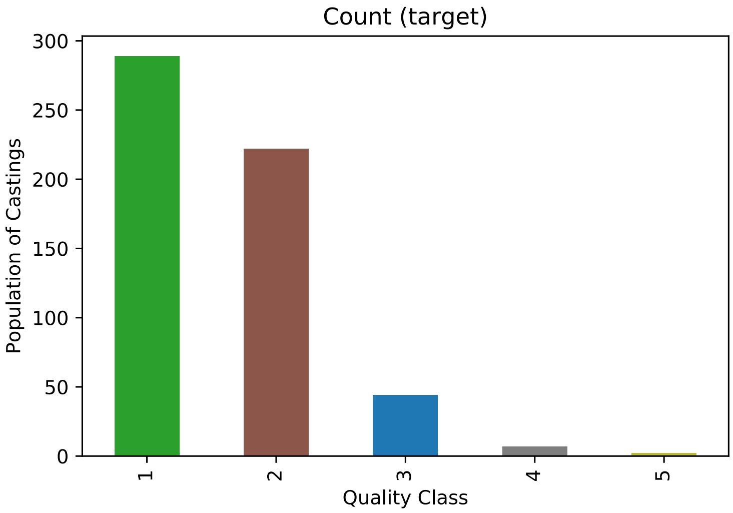


Figure 2.2: Data Imbalance

The total number of items of each class are as follows:

- Class 1: 254
- Class 2: 213
- Class 3: 30
- Class 4: 7
- Class 5: 2

As can be seen, class three, four and five are in the minority. Once again only 30, 7 and 2 samples to learn from which is very less for machine learning to give good precision or recall on these classes.

For this report, in order to be consistent and avoid repetition, we discuss results implemented only on the second dataset.

2.2 Z-Score

The first pre-processing step on the original merged data was to normalize it. We used a Z-score to standardize the data because the various features are measured on widely different scales. This gives us a transformation of the original data that has a mean of zero and standard deviation of 1. The Z-score is defined as given below [14]:

$$Z_x = \frac{X_i - \bar{X}}{S_x} \tag{2.1}$$

where,

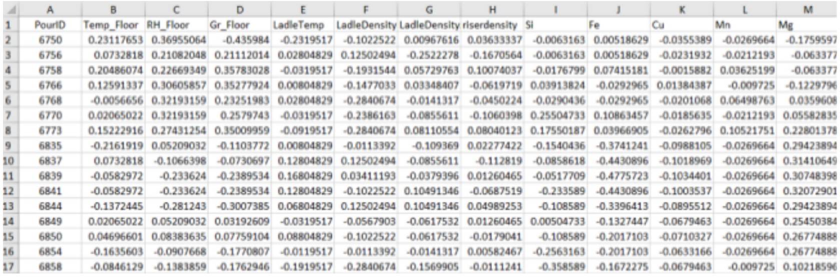
Z_i represents the Z-transformed sample observations, X_i are the original values of the sample, $\bar{X}$ represents the mean of the observations for every column and S_x is the sample standard deviation. The Z-score transforms the data such that it becomes comparable by measuring the observations in multiples of the standard deviation of that sample. This facilitates the interpretation of a single observation [14].

For deep learning models, we also tried other kinds of normalization and scaling methods such as min-max scaling, to see the impact on model performance using different techniques. Min-max scaling is given by:

$$\frac{X_i - X_{min}}{X_{max} - X_{min}} \tag{2.2}$$

Here, X_i refers to the dataframe at hand. The X_{min} and X_{max} refer to the minimum and the maximum values found in every column of the dataset respectively. This helps us restrict the values of the dataset to a fixed range such as between 0 and 1 or -1 and 1, which the Z-transform does not guarantee [15].

The image below shows a snapshot of the dataset, after min-max scaling.



	A	B	C	D	E	F	G	H	I	J	K	L	M
1	PourID	Temp_Floor	RH_Floor	Gr_Floor	LadleTemp	LadleDensity	LadleDensity	riserdensity	Si	Fe	Cu	Mn	Mg
2	6750	0.23117653	0.36955064	-0.435984	-0.2319517	-0.1022522	0.00967616	0.03633337	-0.0063163	0.00518629	-0.0355389	-0.0269664	-0.1759597
3	6756	0.0732818	0.21082048	0.21112014	0.02804829	0.12502494	-0.2522278	-0.1670564	-0.0063163	0.00518629	-0.0231932	-0.0212193	-0.063377
4	6758	0.20486074	0.22669349	0.35783028	-0.0319517	-0.1931544	0.05729763	0.10074037	-0.0176799	0.07415181	-0.0015882	0.03625199	-0.063377
5	6766	0.12591337	0.30605857	0.35277924	0.00804829	-0.1477033	0.03348407	-0.0619719	0.03913824	-0.0292965	0.01384387	-0.009725	-0.1229796
6	6768	-0.0056656	0.32193159	0.23251983	0.02804829	-0.2840674	-0.0141317	-0.0450224	-0.0290436	-0.0292965	-0.0201068	0.06498763	0.0359608
7	6770	0.02065022	0.32193159	0.2579743	-0.0319517	-0.2386163	-0.0855611	-0.1060398	0.25504733	0.10863457	-0.0185635	-0.0212193	0.05582835
8	6773	0.15222916	0.27431254	0.35009959	-0.0919517	-0.2840674	0.08110554	0.08040123	0.17550187	0.03966905	-0.0262796	0.10521751	0.22801378
9	6835	-0.2161919	0.05209032	-0.1103772	0.00804829	-0.0113392	-0.109369	0.02277422	-0.1540436	-0.3741241	-0.0988105	-0.0269664	0.29423894
10	6837	0.0732818	-0.1066398	-0.0730697	0.12804829	0.12502494	-0.0855611	-0.112819	-0.0858618	-0.4430896	-0.1018969	-0.0269664	0.31410649
11	6839	-0.0582972	-0.233624	-0.2389534	0.16804829	0.03411193	-0.0379396	0.01260465	-0.0517709	-0.4775723	-0.1034401	-0.0269664	0.30748398
12	6841	-0.0582972	-0.233624	-0.2389534	0.12804829	-0.1022522	0.10491346	-0.0687519	-0.233589	-0.4430896	-0.1003537	-0.0269664	0.32072901
13	6844	-0.1372445	-0.281243	-0.3007385	0.06804829	0.12502494	0.10491346	0.04989253	-0.108589	-0.3396413	-0.0895512	-0.0269664	0.29423894
14	6849	0.02065022	0.05209032	0.03192609	-0.0319517	-0.0567903	-0.0617532	0.01260465	0.00504733	-0.1327447	-0.0679463	-0.0269664	0.25450384
15	6850	0.04696601	0.08383635	0.07759104	0.08804829	-0.1022522	-0.0617532	-0.0179041	-0.108589	-0.2017103	-0.0710327	-0.0269664	0.26774888
16	6854	-0.1635603	-0.0907668	-0.1770807	-0.0119517	-0.0113392	-0.0141317	0.00582467	-0.2563163	-0.2017103	-0.0633166	-0.0269664	0.26774888
17	6858	-0.0846129	-0.1383859	-0.1762946	-0.1919517	-0.2840674	-0.1569905	-0.0111241	-0.358589	-0.1672275	-0.0679463	-0.009725	0.10218596

Figure 2.3: This figure shows the data after min-max scaling.

2.3 Exploratory Data Analysis

2.3.1 Principal Component Analysis

Many real-world data sets have low intrinsic dimensionality, although being embedded in a high dimensional space. This has made dimensionality reduction, a growing area of research in machine learning. It is important to recover the hidden structure of the dataset. Specific mathematical methods are needed to recover and understand the low dimensional representation of the dataset. [16]

Principal component analysis (PCA) is an unsupervised machine learning method that projects data from a high dimensional space onto a lower dimension subspace. This is helpful so as to plot and understand the data, since human beings cannot visualize data in more than three dimensions. This helps us learn the underlying representation of the data [17]. Compressing the data to a lower dimension can also help save the computation and time for very high dimensional datasets. PCA is a linear transformation of the original dataset. The principal components are linear combinations of the columns of the original dataset. The principal components are chosen in a way such that we do not lose out on information and the direction of the chosen principal components covers the maximum variation of the original dataset in that direction. The first principal component has the highest singular value, meaning that it covers the maximum variation in the data, the second principal component covers the second highest variation in the original dataset and so on [18].

2.3.2 Methodology

In this section, we discuss the steps taken to perform dimensionality reduction using PCA. PCA essentially transforms the original variables in the dataset that are correlated into a smaller number of variables that are uncorrelated. These are known as the principal components. It projects the original data onto a subspace or coordinate system where it maximizes the amount of variance explained in the original dataset [19].

The first and foremost important step before implementing PCA is to normalize the dataset. This is essential because each of the columns of the original data set are measured on different scales. If the transformation is not performed, then the columns that have higher values will be given larger weights and will end up dominating the principal components and contribute the most to the explained variance. We transform the dataset by subtracting the mean of the column and dividing by the standard deviation. It is the same as the Z-score covered earlier. The standardized matrix here will be denoted by Z. This also has additional benefits such as faster learning for neural networks.

The next step is to find a covariance matrix of the dataset. Moreover, the covariance matrix will be calculated on the standardized dataset. This can be done by performing a matrix multiplication of Z^T and Z by normalizing with 1/n-1, here n is the number of features in the dataset. The final matrix obtained after the transformation has n rows and m columns where n and m are the original rows and columns in the dataset [13].

Each of the values on the diagonal of the covariance matrix corresponds to variance explained by each dimension. Every non diagonal element explains the covariance between two features. [20]

The final step would be to perform what is known as the eigen value decomposition, and find the eigen values of the covariance matrix. The final matrix has the eigen values on the diagonal in decreasing order. Here the eigen vector matrix is denoted by V. Moreover, the eigen vector matrix has the columns V_1, V_2, V_3 and so on. These columns represent the first, second and third principal components and so on with the last element representing the last principal component corresponding to the total number of columns in the original data set. Each of the vectors are orthogonal to each other and are ordered in the decreasing order of variance explained.

The steps of the algorithm can be summarized as below:

- Standardize the data. Usually a Z transform as explained earlier, given by subtracting mean and dividing by standard deviation. This is given by Z.
- Find the Covariance Matrix $Z_c = \dfrac{Z^T.Z}{n-1}$.
- Eigen value decomposition which is given by $U \sum V^T$, basically has $\sum$ as the diagonal matrix of interest that carries the eigen values in decreasing order.

  The total number of rows in the data set remains the same as original, only the number of columns in the data set are reduced from M to L after implementing PCA [21].

The below plot shows the two-dimensional PCA plot of the Palmer Foundry data set. It is colored by the quality response variable.

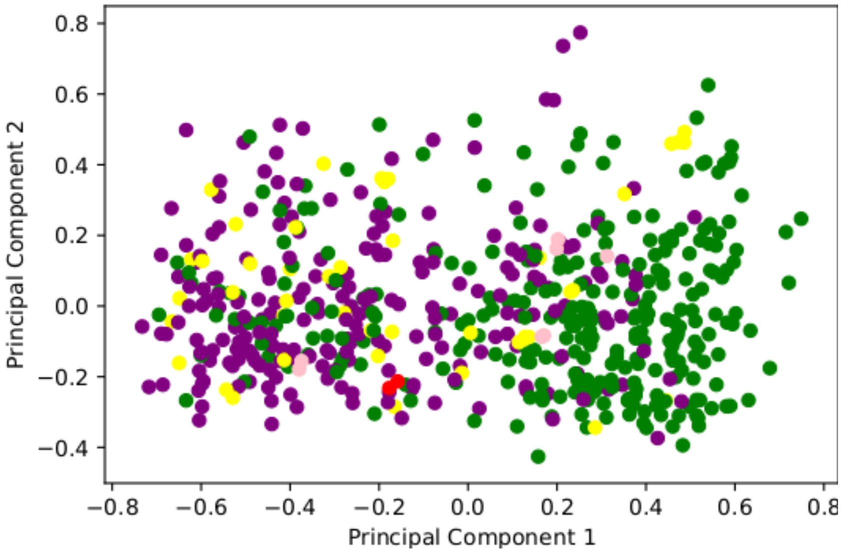


Figure 2.4: Two dimensional PCA plot - Palmer Foundry dataset. It is colored by the quality response column.

The second plot is also called as a scree plot. This plot is used to show the amount of variance that is explained by each of the principal components. The X-axis is the number of principal components and the Y-axis is the cumulative amount of variance

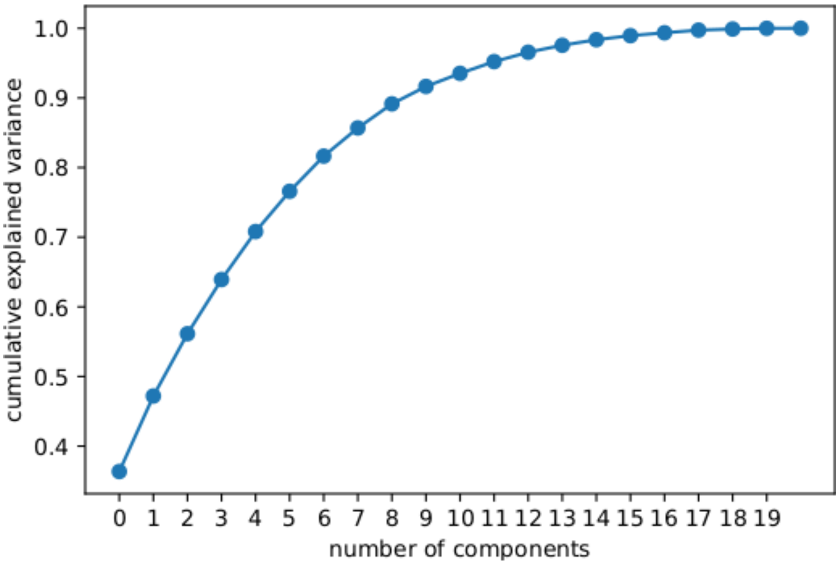


Figure 2.5: Scree plot showing cumulative variance explained using PCA.

explained by each component. The amount of variance explained can also be calculated by the eigenvalue in terms of the ratio of the total sum of the eigenvalues [22].

Plotting and understanding the scree plot is critical, before drawing conclusions about the dataset. As can be seen in the scree plot above, the first two principal components only cover about 50 % variation in the original dataset. This shows that the underlying structure of the data could actually be quite different from what is seen in the compressed representation here. We are losing almost 50% of the variation in the original dataset. Hence, we cannot rely on using the first two components. We also tried using more dimensions, about 8 or 9 principal components, which cover about 85 to 90% variation in the original data, for making predictions using this data with different classifiers. However, the best performance of the classifiers was observed using all the features in the original dataset and without using dimensionality reduction.

A drawback of PCA, is that it only uses linear transformation of the data for dimensionality reduction. However, if the relationship between the features is non-linear, PCA fails to capture the relationships between them. In such a situation, an alternative to this dimensionality reduction is autoencoders. Manifold learning or non-linear dimensionality reduction is essential to understand the intrinsic dimensionality of most real world data sets.

Autoencoders are a deep learning technique that are used to learn to perform non linear transformations using different activation functions. An autoencoder with a linear activation function is essentially similar to PCA [23].

Furthermore, Autoencoders are used to extract features from the data set and the extracted feature vectors can be used for different tasks such as clustering, classification and regression. The raw data is fed to the input layer. The hidden layer of the neural network is used to learn the features of the data set and extract the most useful features that are required to predict the label or output. It captures useful and interesting relations of the input [23].

In this case, it is a regular feed forward neural network where the inputs are the same as the outputs and can be used for optimal representation of the features [21]. The below figure shows the two dimensional comparison of the PCA and Autoencoder models on one of the datasets.

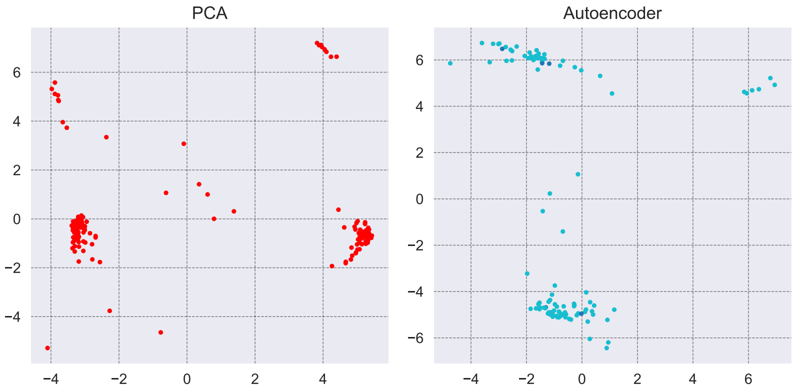


Figure 2.6: This figure shows a comparison between the PCA and Autoencoder techniques for dimensionality reduction.

Autoencoders take an input $X \in R^d$, which is an input in a d-dimensional space, and map it using an encoder to a hidden representation $y \in R^k$ through a mapping, as given by the form: $y = f_\theta(x) = s(W.X^T + b)$. Here s can be any non linear activation function such as a sigmoid or a tanh, depending on the application. In the second part of the network, also known as the decoder, y is mapped back into a reconstruction $\bar{X}$ of the input. Here, $\hat{X} = f_\theta(X) = s(W^T.X + b)$ and it follows a mapping similar to the encoding part of the network. The weights here are updated during training in a way such that the mean reconstruction error is minimized. The loss function is as shown below, which is essentially the frobenius norm of the original and the reconstructed inputs. $L(\hat{X}, X) = \min ||\hat{X} - X||^2$

The hidden layer captures the reduced compressed representation of the input data. We can keep adding multiple hidden layers such that the autoencoder can keep learning multiple hidden compressed representations of the input data. This leads to higher and higher levels of feature representations or feature abstractions.

The lower dimensional representation of a PCA or autoencoder can be used to feed to any classification algorithms to get predictions about the quality or the response variable. The best way to assess which one of these techniques fits the data better, is by plotting it and using it along with a classification algorithm to evaluate the performance.

In case of this dataset, the autoencoder model worked very well for missing data imputation and reconstruction. However, using the hidden layer representation of the autoencoder to make classification predictions did not yield optimal results as compared to the original dataset features. This could also be due to the small size of the dataset and we need more data to confirm the results.

It is important to note that, it is not necessary that an autoencoder will always outperform a simpler technique like the PCA. Moreover, if a linear transformation is good enough to learn the dataset at hand, without any loss of information, then a complicated deep learning network like an autoencoder may not be necessary [21]. The chapter on missing data imputation discusses the implementation on autoencoders in depth.

2.3.3 Feature Selection

Often in data science we have hundreds or even millions of features and we want a way to create a model that only includes the most important features. This has mainly three benefits. First, we make our model simpler to interpret by reducing the number of variables included in the analysis. Second, we can reduce the variance of the model, and therefore over fitting. Finally, we can reduce the computational cost (and time) of training a model. The process of identifying only the most relevant features is called feature selection.

The Figure 2.7 shows feature importance using an algorithm known as XGBOOST, short for extreme gradient boosting. Xgboost is a library for gradient boosting algorithms. Boosting algorithms work on the principle of using multiple weak learners and converting them to strong learners. It is an iterative technique where the model works in a sequential manner by adjusting weights of the weak learners as training progresses. This is effective in terms of bias reduction and improvement of accuracy [24] . The plot 2.7 shows the important features in the dataset using this technique, in terms of the f-score. The f-score here indicates the number of times that a particular feature is used at a split in the decision tree. The more the number of times a feature is used, the more important it is for predictions.

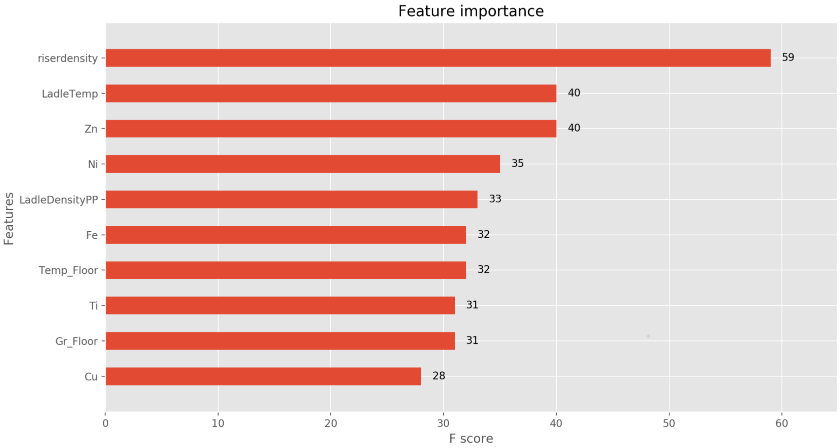


Figure 2.7: This figure shows the important features in the dataset using the XGBoost algorithm and a f-score criterion.

The feature importance score using a combination of different algorithms is seen in Figure 2.8. This is a more robust approach as compared to using only Random Forest or any other algorithm by itself, so that we can verify the results obtained using multiple algorithms to see if they are in agreement. This is especially true when the size of the dataset is extremely small, such as this one.

Feature Engineering is one of the most critical pre-processing steps for implementing effective machine learning. Feeding the algorithm with the right information is crucial. Choosing the right parameters of the data set and taking into account multiple factors such as the composition of the alloys, environmental conditions in the foundry, changes

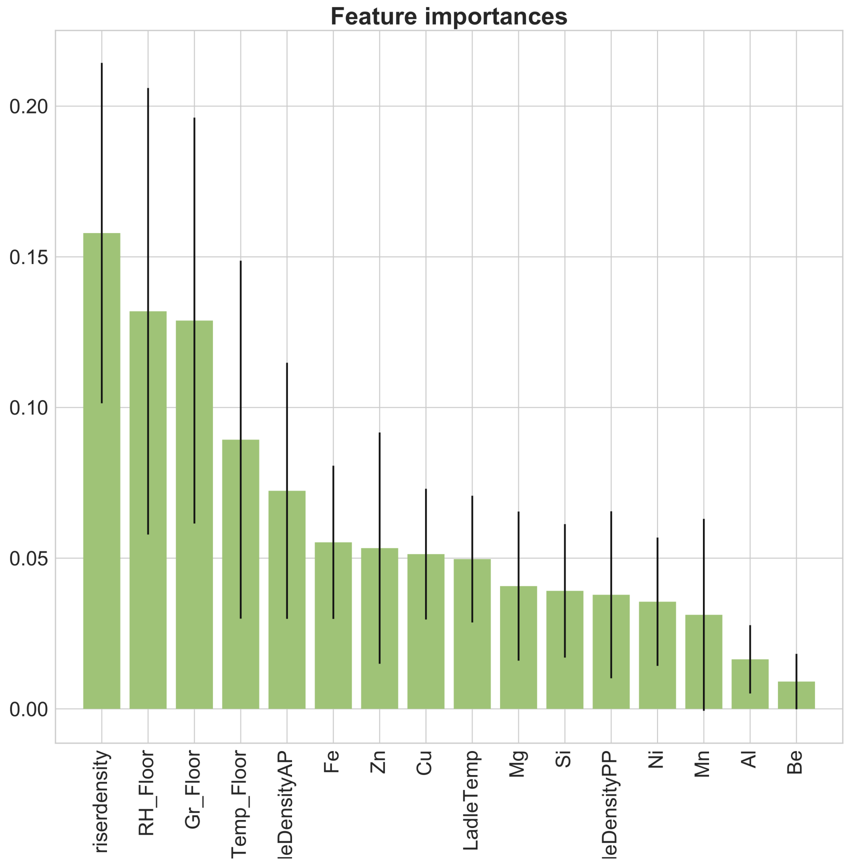


Figure 2.8: Feature Importance using a combination of techniques such as Random Forest and SVC is shown below. It works by finding the most important features individually and then taking a majority vote to find the final features that are important.

in temperature during the entire process etc. and deciding which of these are most important in terms of quality prediction is crucial for effective process control. Feeding the algorithm with the right information and communicating the results to the foundry is important for improving their process cognition.

There are a number of decisions to be made, such as:

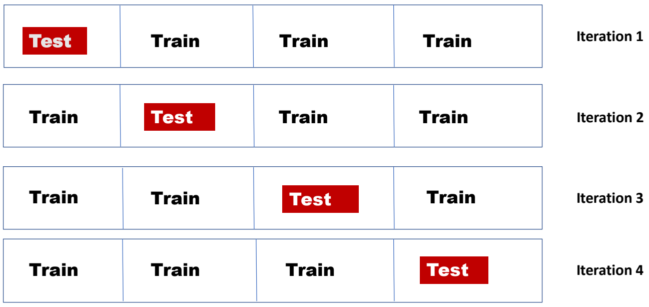
- How should the chemical composition data be represented to be comparable with the environmental and temperature data?
- How frequently should the temperature be measured?
- Is providing the ambient humidity important?

Some of these decisions are relatively easier to make as compared to the others. Feature selection is important to link the features together and help in answering such questions. The above techniques help select the features that are most important to answer the question at hand.

In this study, feature selection techniques were used for dimensionality reduction before implementing classification algorithms. However, across different algorithms, overall, it was observed that feature selection did not significantly improve performance as compared to using the original features in the dataset for classification. [1]

2.4 Cross Validation

Cross Validation is an important step before implementing any machine learning. Splitting the data set into training, testing and validation is an effective way to improve the accuracy of predictions and can be used while evaluating the performance of the algorithms. The algorithm needs to be tested in a manner that is faithful to how it will be used in a real world scenario. Cross Validation used for the datasets in this study was K-fold and stratified K-fold, as explained below:



Test	Train	Train	Train	Iteration 1
Train	Test	Train	Train	Iteration 2
Train	Train	Test	Train	Iteration 3
Train	Train	Train	Test	Iteration 4

Figure 2.9: K-fold Cross Validation [4].

Cross Validation is a common practice used for machine learning, to prevent over fitting of a training dataset. It is important that the classifier performs well on data that it has not seen earlier and has a good generalization performance on parts that are being rolled off of an assembly line. The idea is to divide the entire data into multiple folds. We train the data set on K-1 out of the K folds and validate it on the Kth fold. This technique was used for hyper parameter tuning of the algorithms. For example, in a Random Forest algorithm, we need to find the optimal number of trees to be used in the Forest, the optimal number of parameters to be used or considered while splitting at each node of the tree.

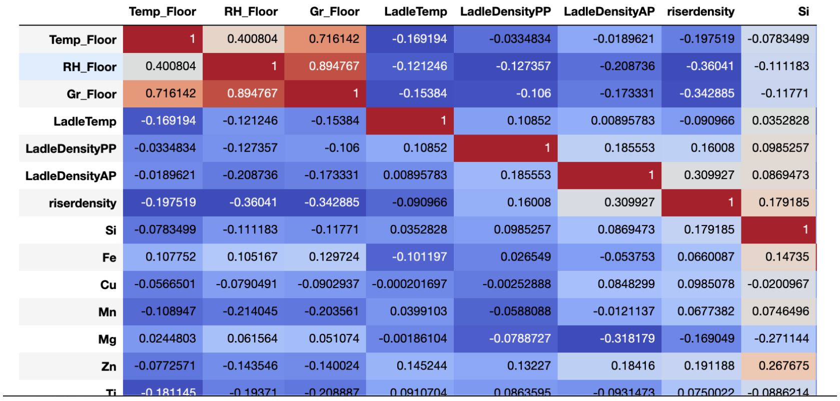
We have done this using K-fold cross validation. Each time, a different fold out of the K folds is used for validation and the remaining K-1 folds are used for training. We can do this K times, which is usually 5 or 10 [25]. This way, we can change the parameters while training and validating multiple times on different folds, in order to find the parameters that give us the best performance in terms of the metric that we are trying to maximize. Once we find these optimal parameters using cross validation or a grid search, we can use these same set of parameters to make predictions about new unseen testing data. For this study, we used a similar approach. The data was split as 70 % into training and validation and 30 % as testing data.

Another approach known as stratified K-fold cross validation was also implemented because the dataset is highly imbalanced. Stratified K-fold is a variation of K-fold cross validation in which the folds are split in such a way that the ratio of the classes is representative of that in the entire data set. For example, in a binary classification problem, if a data set has roughly half instances of each of the classes, then every fold in the K-folds will have roughly half instances of each class. Stratified K-fold is therefore better in case of data sets that are unbalanced. Owing to this reason, stratified sampling gave better results on the imbalanced classification schemes, as compared to using only regular cross validation. Using cross validation, we test the performance on multiple subsets of the data and take the average of the metric across all the folds as the final measure of performance. We used a number of different metrics includingF_1score, precision and recall to measure performance [26]. We have used both, K-fold cross validation as well as stratified K-fold cross validation in this study and the further sections would specify the technique used for different algorithms.

2.5 Pearson Correlation

The below table shows a correlation matrix heatmap of the dataset. This is Pearson's Correlation matrix. The Pearson product moment correlation coefficient shows the strength of the linear relationship between any two quantitative variables. However, this is only for linear relationship between variables. If it is not a linear relationship, then this coefficient cannot be used or rather does not strongly represent the relationship between the two variables [27]. This is denoted by r and in order to understand whether this can be used as a measure or not, can be decided based on whether the two continuous variables or the relationship between the response and the predictors is a linear one. This can be seen by using scatter plots and using the correlation coefficient if a linear relationship is seen between the two variables. The response variable is usually drawn on the y-axis where as the predictor is on the x-axis. The nearer the scatter points are to a straight line the better the strength of the linear relationship is and higher the association between the variables is. This is irrespective of the measurement units that are used in the data. It can be seen that features like RH_Floor and Gr_Floor are highly correlated. In such situations, we can eliminate one of the two highly correlated features and then perform analysis. This is owing to the fact that we are not adding value to the analysis by including both the features that are highly correlated. We used either one of RH_Floor and Gr_Floor during classification. It was seen that eliminating one of these features slightly improved the classification performance of the algorithms, although not significantly [27].

The Figure 2.11 image shows the distribution of the features in the Palmer foundry dataset . We can check the amount of overlap in the distributions of the good and the bad classes in order to find the features that are the most important in terms of quality prediction. These show the marginal contributions of the features to the part quality. Although there is a lot of overlap and we cannot readily spot the differences, we can still identify a few trends such as, for example, values of riser density below a certain threshold clearly increases the fraction of bad quality parts, whereas, above the threshold, increases the fraction of good quality parts. It is critical that the foundry maintains the values of the variables in the range that reduces the fraction of bad parts.



	Temp_Floor	RH_Floor	Gr_Floor	LadleTemp	LadleDensityPP	LadleDensityAP	riserdensity	Si
Temp_Floor	1	0.400804	0.716142	-0.169194	-0.0334834	-0.0189621	-0.197519	-0.0783499
RH_Floor	0.400804	1	0.894767	-0.121246	-0.127357	-0.208736	-0.36041	-0.111183
Gr_Floor	0.716142	0.894767	1	-0.15384	-0.106	-0.173331	-0.342885	-0.11771
LadleTemp	-0.169194	-0.121246	-0.15384	1	0.10852	0.00895783	-0.090966	0.0352828
LadleDensityPP	-0.0334834	-0.127357	-0.106	0.10852	1	0.185553	0.16008	0.0985257
LadleDensityAP	-0.0189621	-0.208736	-0.173331	0.00895783	0.185553	1	0.309927	0.0869473
riserdensity	-0.197519	-0.36041	-0.342885	-0.090966	0.16008	0.309927	1	0.179185
Si	-0.0783499	-0.111183	-0.11771	0.0352828	0.0985257	0.0869473	0.179185	1
Fe	0.107752	0.105167	0.129724	-0.101197	0.026549	-0.053753	0.0660087	0.14735
Cu	-0.0566501	-0.0790491	-0.0902937	-0.000201697	-0.00252888	0.0848299	0.0985078	-0.0200967
Mn	-0.108947	-0.214045	-0.203561	0.0399103	-0.0588088	-0.0121137	0.0677382	0.0746496
Mg	0.0244803	0.061564	0.051074	-0.00186104	-0.0788727	-0.318179	-0.169049	-0.271144
Zn	-0.0772571	-0.143546	-0.140024	0.145244	0.13227	0.18416	0.191188	0.267675
Ti	-0.181145	-0.19371	-0.208887	0.0910704	0.0863595	-0.0931473	0.0750022	-0.0886214

Figure 2.10: This figure shows the correlation matrix based on Pearson's correlation between variables in the dataset. This shows the strength of linear relationship between the features of the dataset.

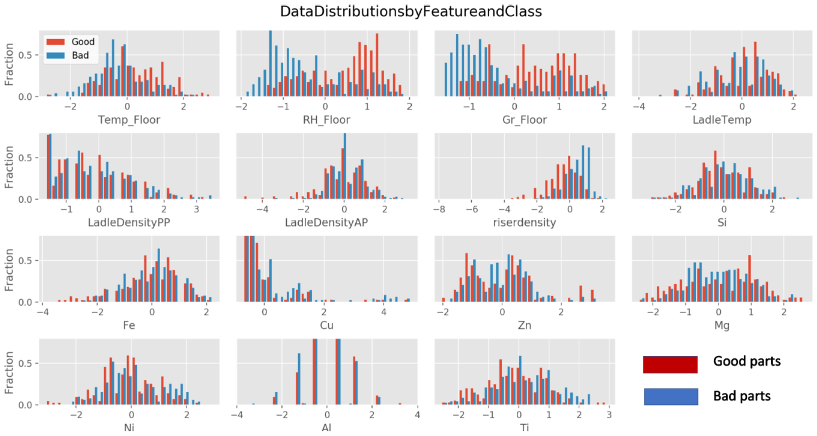


Figure 2.11: This figure shows the marginal distributions of the features in the dataset in terms of the quality. It can be seen that some of the features are more important in terms of separating the good and the bad quality parts as compared to the others.

Chapter 3

Methods for treating small, semi-supervised and imbalanced data

This chapter provides information on the resampling techniques and other approaches that were used in this study to deal with the small sized and imbalanced nature of the problem. These techniques are a combination of over-sampling and under-sampling techniques as well as cost sensitive learning. As mentioned earlier, deep learning approaches such as generative adversarial networks (GANs) have also been used in this study for data augmentation. However, the semi-supervised GAN shows optimal performance when the dataset is balanced. The minority classes, in particular, classes 3, 4 and 5 have very few samples for the network to learn the underlying structure of the data from, and make predictions on the unseen test data.

Moreover, as will be discussed in the section on GANS, the semi-supervised GAN outperforms other classifiers when we use a binary and balanced one-vs-all classification scheme, where class 1 refers to good quality parts and classes 2, 3, 4 and 5 are all combined to form the bad part quality class. We show a comparison in terms of the F_1score using the GAN along with other classifiers. The GAN has a higher median value in the box plot and it is also more consistent across multiple folds of cross validation as compared to any of the other classifiers. The sections below detail each of the techniques that were implemented.

3.1 Random oversampling and undersampling

We used random oversampling and undersampling techniques to balance the class proportions. However, in random oversampling, the algorithm has a high potential to overfit because it does not generate additional synthetic datapoints, but only duplicates the existing data points. Similarly, random undersampling has a drawback that while randomly downsampling the majority class, we could be throwing away important information in the dataset. Owing to these reasons, we implement the SMOTE technique, which will be described below [12].

3.2 Synthetic Minority Oversampling(SMOTE)

An example of the SMOTE balancing technique two dimensional PCA plot for the Palmer foundry dataset is as shown below [12]:

The SMOTE algorithm generates synthetic samples of the minority class, such that they are equal to or nearly equal to the number of samples of the majority class. It takes an instance of the minority class and randomly chooses from one of its nearest neighbors and takes a linear combination of these to generate synthetic data. We can manually change the number of neighbors that it considers while interpolating between two points of the minority class.

However, SMOTE has a drawback such that it reduces the variance in between the samples and introduces correlation between them. This could impact the classification algorithms that depend on the assumption of independence of samples. Most of the variable selection methods assume that the samples are independent [28].

The below equation can be used to describe the SMOTE technique.

$$s = Z + u \times (Z^S - Z), \quad with \;\; 0 \leq u \leq 1 \tag{3.1}$$

Here Z^S is amongst the nearest neighbors of the minority class sample under consideration. The number of nearest neighbors to be used while generating synthetic data for

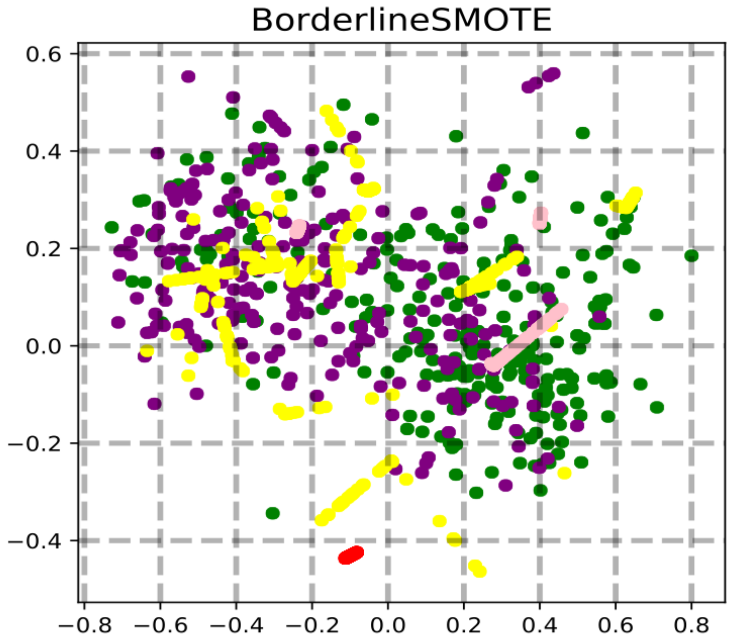


Figure 3.1: SMOTE two dimensional, colored by quality

a particular minority class sample can be changed and the best number can be selected for a particular dataset.

We also implemented variations of SMOTE such as borderline SMOTE, in which we focus on the borderline examples of the majority and the minority class while resampling because the samples at the borderline are more prone to misclassification errors as compared to the ones away from the border.

3.3 Generative Adversarial Networks(GAN)

Generative adversarial networks(GANS) are a popular deep learning technique used for data augmentation. This set of networks originally belong to the field of unsupervised learning, as they do not require the labels or response variable for the generation of the synthetic data. They are generative models that learn the underlying distribution or structure of the data without specifying the target value. GANs learn the intrinsic distribution of the classes in the data set, that has multiple classes, such that they generate the synthetic data for all the classes that belong to the original dataset, so long as we have large number of samples belonging to each of the classes [5].

3.3.1 Evaluation Metrics

The two most commonly used metrics for the comparison of GAN performance, are the Kullback-Leibler Divergence(KL divergence) and the Jensen Shannon Divergence(JS divergence). The KL divergence is essentially a measure of how one probability distribution P diverges from the other Q. It is given by the following:

$$KL(P||Q) = -\sum P \log(P)/\log(Q), \tag{3.2}$$

The KL divergence is not symmetric, whereas, the JS divergence is symmetric. The places where the distribution of P is zero and Q is non-zero, the effect gets disregarded. This could be problematic when we have two distributions that we are trying to compare where both are equally important. It is important that for the regions where P has a non null mass, Q also has a non null mass. There is an integral term in the equation for the KL divergence which explains that if the distribution Q is chosen to minimize the KL metric, it is unlikely that Q will assign a lot of mass to regions where P is close to zero. JS Divergence on the other hand behaves in a similar manner for small values of P or Q [29].

The architecture of a GAN is as summarized below:

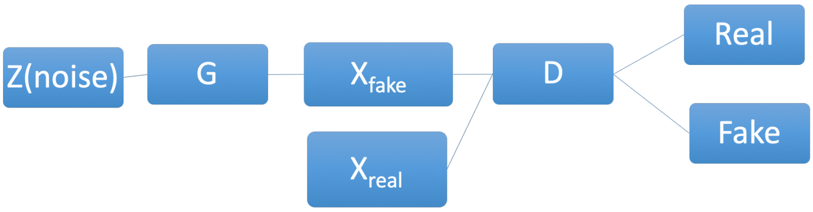


Figure 3.2: This figure shows the architecture of a vanilla GAN model [5].

It has two neural networks working against each other, namely the generating network and the discriminating network. The discriminator D has the objective of identifying the images as real or fake. The adversarial generating network has the goal of tricking the discriminating network into believing that the fake images are real. During training, the discriminator is trained on random images of real data along with an equal proportion of images generated by the generator. The task of the discriminator is to distinguish between the real and the synthetic images. It outputs a probability between 0 and 1, where 0 meaning that the image is identified as fake, 1 meaning that the discriminator thinks that the image is real. Anything in between gives us a probability that the image is real. Depending on the outcome of the discriminator, both the networks try to optimize their own parameters by fine tuning their networks and becoming better at their objective. If the discriminator is easily able to discriminate between the real and the generated images, the generator does a better job creating artificial images, making it harder for the discriminator to be able to distinguish between them, in the next iteration. After training for a sufficiently long period of time, there would come a time when the discriminator outputs a probability of 0.5, meaning that it is no longer able to distinguish between the real and synthetic data. This would be the ideal situation, meaning that the generator is generating very realistic looking data, such that the discriminator is not able to tell it apart from the real data. At this point, both the networks cannot improve anymore and have converged.

This is a mini-max game with the value function $V(G, D)$ which is given below:

$$\min G, \max D \quad V(D,G) = \mathbb{E}_{x \sim p_{data}}[\log D(x)] + \mathbb{E}_{z \sim p_z(Z)}[\log(1 - D(G(Z)))] \tag{3.3}$$

It has two loops, the outer loop is trying to minimize the equation with respect to the generators parameters alone and the inner loop is trying to maximize the equation with respect to the discriminator's parameters alone. It can be seen, based on the log values, that the two networks are in an adversarial mode where they have opposing tasks in the game which they try to satisfy until convergence [30].

Furthermore, if the discriminator is not able to classify the real and synthetic images, it will update its parameters in order to do a better job in the next iteration. The total reward for the discriminator is the total number of correct predictions that it makes and the reward for the generator is the total number of discriminator's errors. This process continues until the parameters are optimized and equilibrium is achieved. Moreover, the discriminator weights are updated in such a manner that they maximize the probability of a real data sample x being classified as belonging to the real data set. On the other hand, the discriminator minimizes the probability that a fake sample is classified as belonging to the real data set. The loss or error function used maximizes the function $D(x)$ and also minimizes the $D\Big(G(z)\Big)$.The log probability is used in the loss functions instead of raw probabilities, since a log loss heavily penalizes the incorrect classifications of an algorithm that is confident about its predictions. This can be seen in the log graph as shown below.

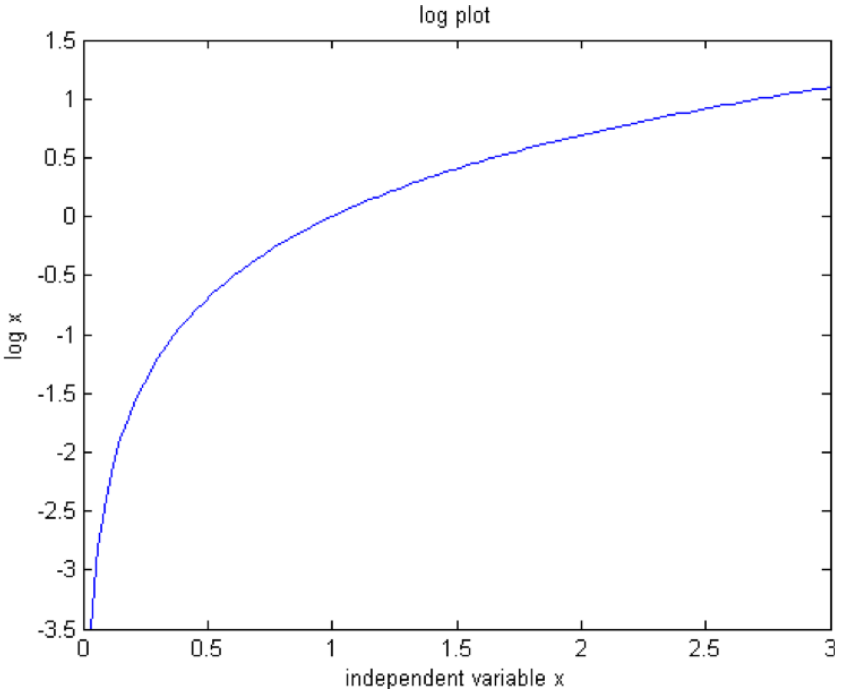


Figure 3.3: Log plot [6].

The generators distribution $P(g)$ is given over data $X.D(x;\theta)$ outputs a single scalar. $D(x)$ is the discriminating network and it gives a probability value between 0 and 1 expressing the probability that x is coming from the original data and not the generating network. The generating network G is trained to minimize $\log\Big(1 - D(G(Z))\Big)$. The discriminator loss is given by Eq.(3.4):

$$\mathcal{L}_D = \frac{-\mathbb{E}_{x\sim p_{data}} \log[(D(x))] - \mathbb{E}_{z\sim p_z(Z)}[\log(1 - D(G(Z)))]}{2} \tag{3.4}$$

The generator loss is given by the Eq.(3.5):

$$\begin{aligned} \mathcal{L}_G &= \mathbb{E}[-\log{(D(G(Z))]} \\ \mathcal{L}_G &= \mathbb{E}[\log{(1 - D(G(Z)))]} \end{aligned} \tag{3.5}$$

The generator network tries to maximize the probability that the fake sample is classified as real by the discriminator network. One of the biggest issues with training a GAN is not having sufficient training data. It is very expensive and tedious to collect data. This is especially true in the metal casting context. This makes it difficult for the GAN to learn the underlying structure of the data from only a few examples. There are very few bad quality parts in the metal casting data set. This makes it challenging for the GAN to learn the underlying structure of the data from very few data points, unless the dataset is balanced.

Generative models can also be used for classification tasks, where the discriminator distinguishes between real and fake data and also outputs the probability of the class of the data, if it is a real sample. This is a part of a semi supervised learning architecture of a GAN as mentioned earlier. The architecture of the semi-supervised GAN is shown in the figure 4.11. The architecture of a semi-supervised GAN is as shown below:

Generative models try to model the real data distribution such that the real samples have as high a probability as possible. Maximum Likelihood estimation is used for the same purpose. Maximizing the maximum likelihood estimation is same as minimizing the KL divergence metric mentioned above. The loss of the generator can be given as the negative of the loss of the discriminator $L(G) = -L(D)$.

If the discriminator is trained on an equal number of samples of the generated data and the real data, the loss function or the expected absolute error of the discriminator will be given as below:

$$E(G, D) = \frac{1}{2}\mathbb{E}_{x\sim p_x}[1 - D(x)] + \frac{1}{2}\mathbb{E}_{z\sim p_z}[D(G(Z))] \tag{3.6}$$

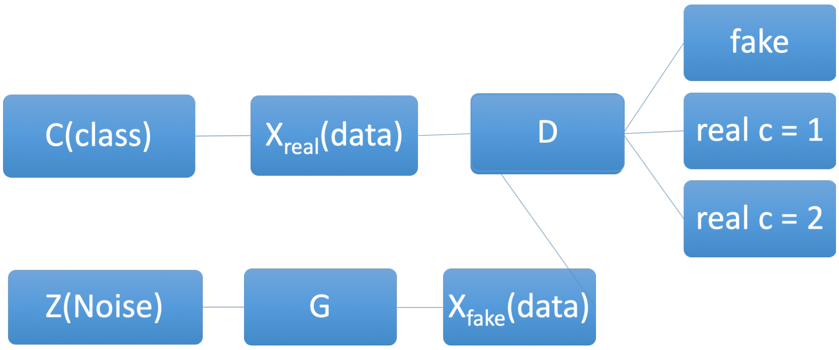


Figure 3.4: This figure shows the architecture of a semi-supervised GAN. It can be seen that this is a variation of a regular GAN in which along with the real and fake problem, the discriminator can also be used as a supervised classifier, where we have K classes, representing the real data and an additional class is added for the data coming from the generator [7].

The ultimate goal of the generator is to trick the discriminator and try and make the discriminator output real for fake images and vice versa. Hence, when training the generator, we try to maximize, the same error metric for the generator that we try to reduce for the discriminator. Finally, for a good performing GAN, the equation that a discriminator is trying to minimize is the one given above. The details of the semi-supervised network will be detailed in the next section.

3.4 Semi Supervised GAN

GANs are generative models that are known to work very well in case of high dimensional probability distributions, which is the case for many real world data applications. In the metal casting context, GANS have proven to be effective in semi supervised learning scenarios, where we have a mix of both labelled as well as unlabelled data points. GANS model different modes of a data set and learn multiple modes for data generation. This is also useful for example in cases where one instance in a data set may have multiple classes [5].

The purpose of using the architecture shown above is two fold. It can be used as a classifier and eliminates the need for a separate classifier. It also is proven to produce better quality data as compared to a regular GAN, in a shorter period of time. Here we have the generator G and discriminator D which is also the classifier C. The three networks are working in co-ordination with each other and the feedback from one network, improvises the performance of the other. The discriminator feed forward layer final output layer has one single sigmoid unit that indicates whether a particular sample is drawn from a data generating distribution. We use softmax as the final layer unit activation for getting the class probabilities in case of multi-label datasets. The discriminator will have in this case, K+1 output units, where we have K corresponding to the number of classes in the training data and one additional class is added for the synthetic samples coming from the generator. [7]

The discriminator/classifier network are trained to reduce the negative log likelihood of the samples *w.r.t.* their labels and on the other hand, the generator is trained to maximize it. Different weight parameter initializations and other architectures are compared to see the quality of the fake data that is generated by these generative models.

Different parameters, in terms of the batch size, learning rates, activation functions and optimizers were tried for the semi-supervised GAN model using cross validation. The best performance in terms of a weighted average F_1score for both the classes, was obtained using Adam optimization, and using a learning rate of 0.002 and an average batch size of 200. The activations used were Relu and LeakyRely except for Tanh in the last layer of the generator. This was done because we wanted data to be generated between -1 and 1, same as our original min-max scaled data.

This type of classification scheme takes a small portion of the labeled data and a much larger portion of unlabeled data. We can specify the amount of unlabeled data that we want to use to see how the performance changes as the percentage of labeled data for training changes.

For the regular GAN problem, the discriminator tries to maximize the probability of the fake samples to be zero and for the real samples, it tries to maximize their probability close to 1. Depending on the feedback of the discriminator, the generator updates its parameters in such a way that it improves and produces more realistic looking samples in the next epoch.

The discriminator is trained on equal size samples of the real and the generated data , corresponding to K+1 classes, where the K classes are those from the real data and one is added for the generated data. We can then use it for classification on the real testing data. The softmax output gives the probabilities of the classes. To get the predictions, we use the maximum probability of all the real classes and exclude the last class which corresponds to the synthetic data.

Now, the generator is trained on the outputs of the discriminator with only the generated data of size d until convergence. To summarize the above, the tasks of the discriminator are as follows:

- Help the generator produce realistic looking images.
- Use the generator's images along with the labeled and unlabeled data, to help with the classification of the real test dataset.

The sources of training data to the discriminator are as given below:

- The real data with labels
- The real data without labels
- The fake data generated by the generator.

For the labeled data, the discriminator learns to classify it like any regular supervised learning problem.

For unlabeled real data, the discriminator learns only that it is real. The unlabeled data from the generator, the discriminator only learns that it is synthetic. The combination of these different sources helps the classifier perform inference from a much broader perspective. Extensive use of regularization and dropout is made in order to prevent over fitting and help with good generalization on unseen examples [31].

3.5 Model loss for semi-supervised GAN

Refer to figure 3.10 below which shows loss function of Semi supervised GAN discriminator, which is a sum of the supervised as well as unsupervised losses:

$$\mathbb{L}_D = \mathbb{L}_{supervised} + \mathbb{L}_{unsupervised}, \tag{3.7}$$

$$\mathbb{L}_{supervised} = -\mathbb{E}_{x,y \sim p_{data}} \log \mathbf{D}(y|x, y < K+1), \tag{3.8}$$

$$\mathbb{L}_{unsupervised} = -\left\{ \mathbb{E}_{x \sim p_{data}} \log \left[1 - \mathbf{D}(y = K+1|x) \right] + \mathbb{E}_{z \sim p_z} \log \left[\mathbf{D}(y = K+1|G(Z)) \right] \right\}. \tag{3.9}$$

Here G(Z) is a sample that belongs to the generated data distribution, whereas x is a sample that comes from the real data distribution. This is essentially composed of two parts, the sigmoid cross entropy loss function, which is used to compute the loss for the real vs fake regular GAN problem. For real data coming from the training set, we maximize the probabilities of being real by giving them labels of $1s$. For fake data coming from the generator, we maximize the probability of being fake by assigning them labels of $0s$. For the supervised loss, multi-class classification problem, we use the softmax cross entropy function with real data labels, that we have available. Although we use equal weightage on the supervised as well as unsupervised losses in this study, we would like to use a hyperparameter lambda in the future to adjust the weights of both the losses to see how it impacts performance.

In summary, the discriminator loss is a sum of the supervised and the unsupervised losses, namely the sigmoid cross entropy and the categorical cross entropy loss. We use the labeled data for training and the unlabeled data for testing. We also use a mask variable to see the impact of masking some of the labeled data and as expected, with decrease in the % of labeled data, the classification performance goes down. We use binary cross entropy loss for the generator as well as discriminator since both the real vs fake as well as supervised classification can be considered as binary classification problems in this case.

This study shows that GANS are a helpful set of models to learn complex tasks when we have less labeled data available [7], that we can leverage and learn from along with the labeled data. We use cross validation on both the generator and the discriminator

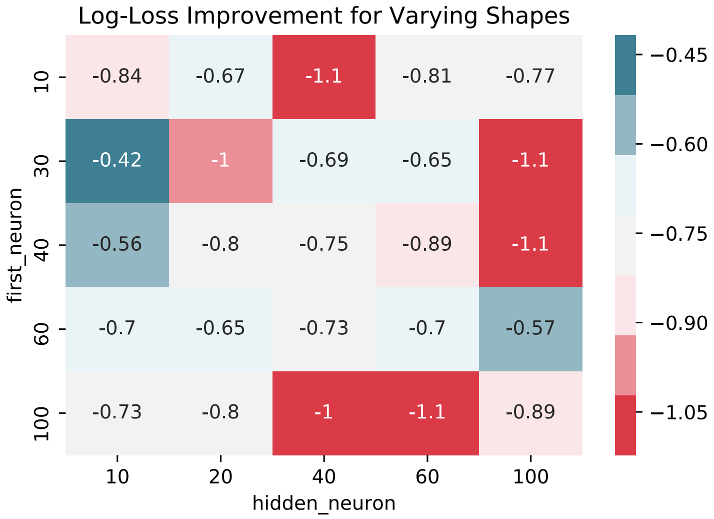


Figure 3.5: Hyperparameter Tuning

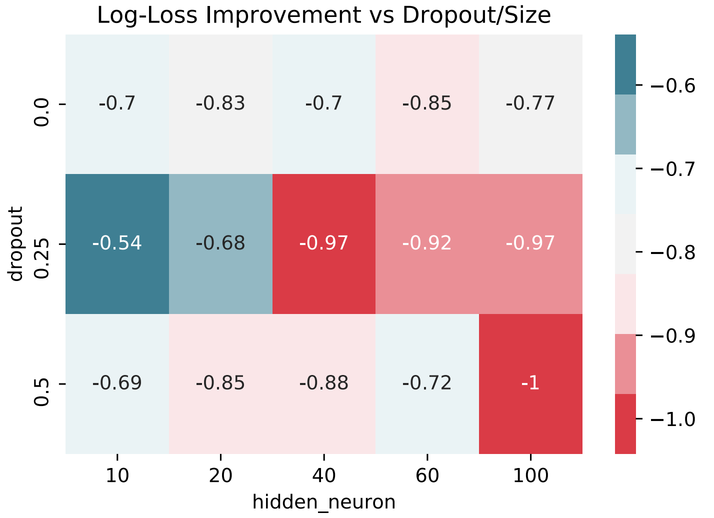


Figure 3.6: HyperparameterTuning

networks using a binary cross entropy or log loss improvement metric, to find the optimal parameter of both the networks. The below plots show an example of hyperparameter tuning for the GAN discriminator in terms of its loss in discriminating between real and generated data.

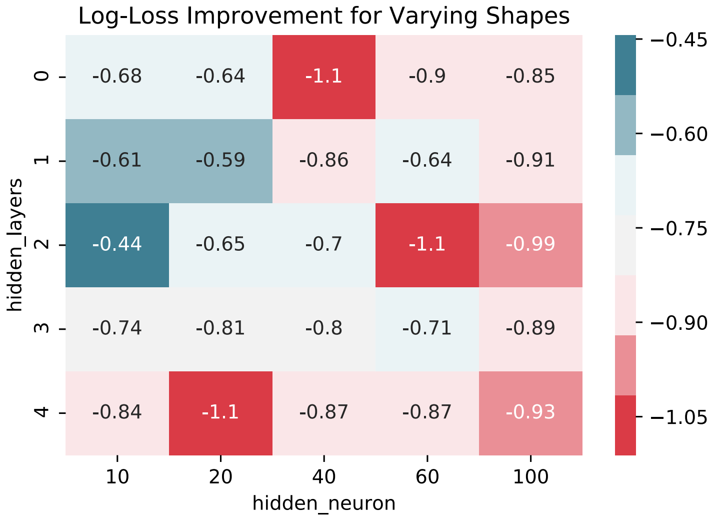


Figure 3.7: Hyperparameter Tuning

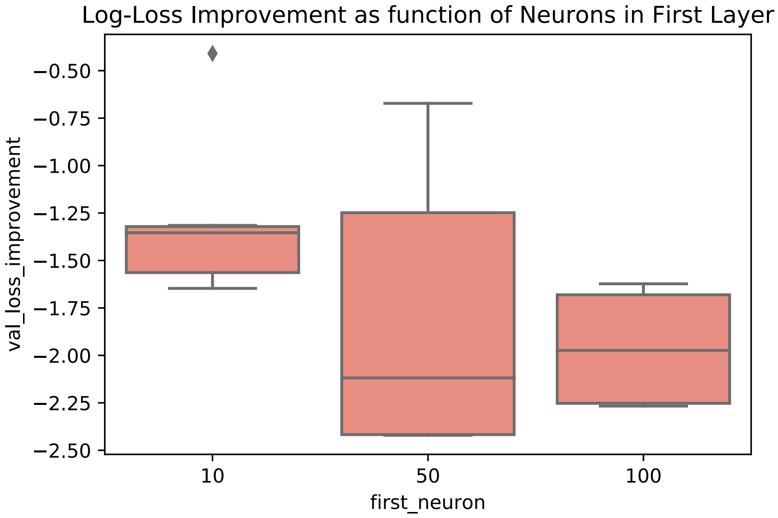


Figure 3.8: Hyperparameter Tuning

In one of the novelties of this study, we also showed that GANS show a potential to overfit in this setting. The plot below shows the training and testing accuracy as well as loss. It can be seen that as the training progresses, the GAN starts generating the same data points, which leads to increase in the accuracy of the training data, however, leads

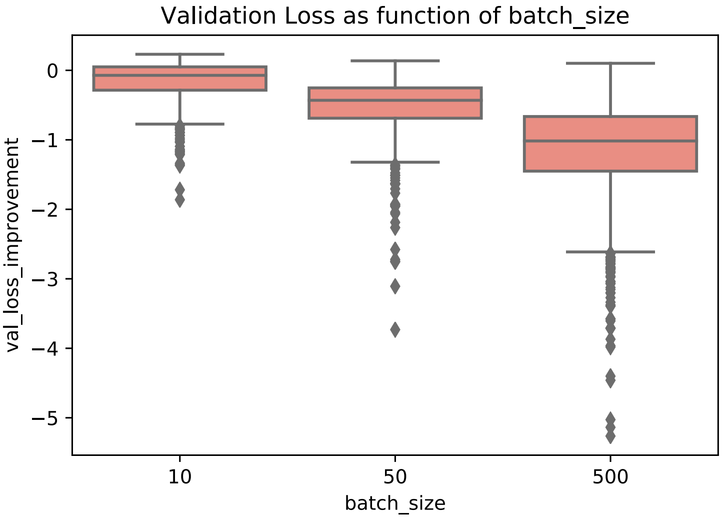


Figure 3.9: Hyperparameter Tuning

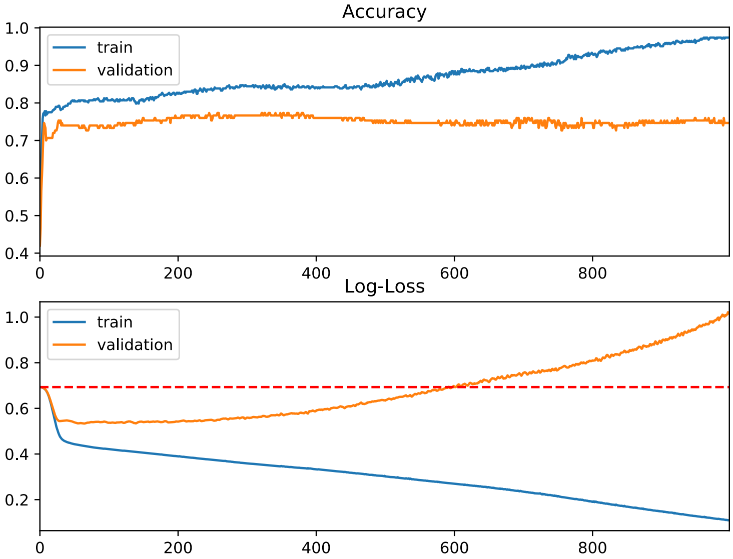


Figure 3.10: GAN overfitting across multiple epochs of training

to reduction in the testing data accuracy due to overfitting. Owing to this, we perform in depth hyperparameter tuning using cross-validation and test the performance of the network on changing architectures.

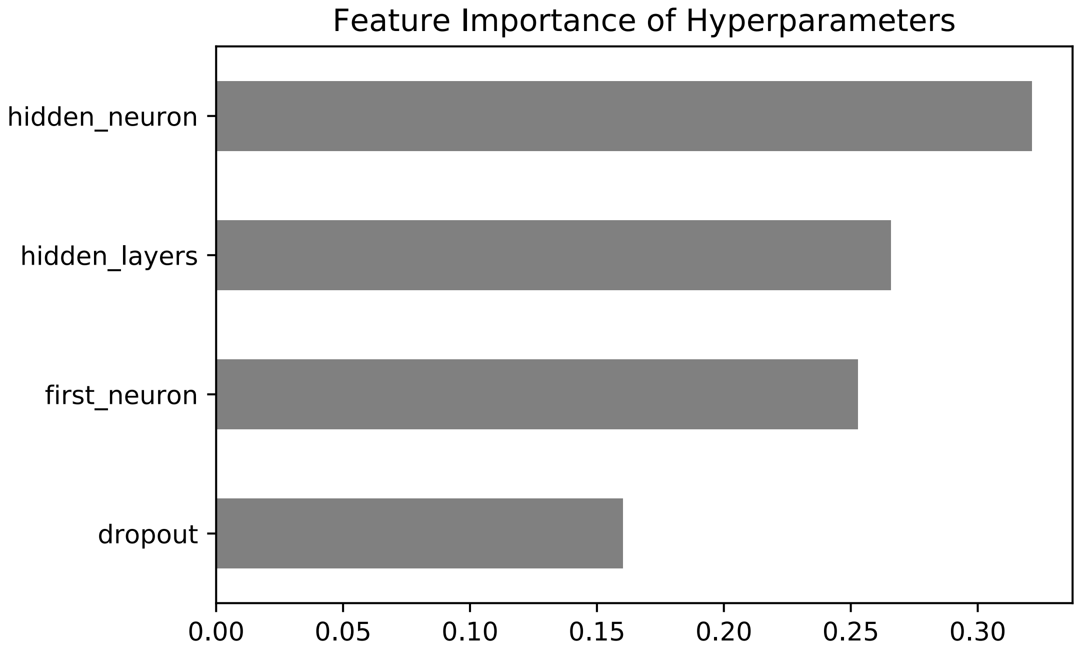


Figure 3.11: This figure shows the most important features in terms of the validation loss improvement using a Random Forest feature importance technique. It can be seen that the number of nodes in the hidden layer is the most important parameter followed by the number of hidden layers in the network.

We use early stopping criterion and stop training when the loss on training starts increasing, however, starts decreasing on the validation set. We do this multiple times on different cross validation folds, in order to achieve optimal performance. The higher the value of log loss improvement, the better the parameter. Overall, we observe that shallower and smaller networks are more suitable for this dataset, as compared to deeper or larger networks. This is reasonable to interpret, given the small size of the dataset. The results obtained for the supervised classification problem is compared in the box plots below with other classifiers in terms of the weighted average of the F_1score metric. It can be seen that the the GAN is not only the highest but also the most consistent as compared to any of the other classifiers. Figure 3.11 shows the importance of the parameters using a Random Forest algorithm to find the most important features in terms of validation loss improvement. It can be seen that the number of nodes in the hidden layer is the most important feature followed by number of hidden layers.

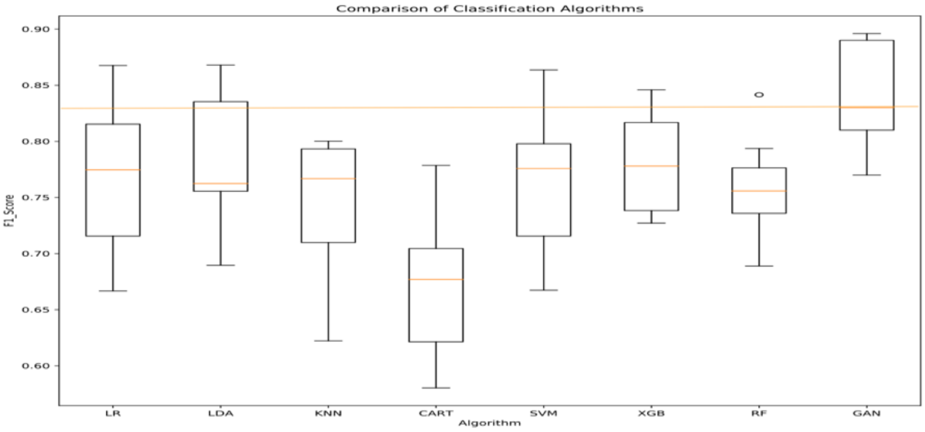


Figure 3.12: Comparison of classifiers in terms of the F_1score. It can be seen that the GAN model gives the best and most consistent performance in terms of the weighted average of the F_1score on a balanced binary classification dataset.

Chapter 4

Missing data imputation of heterogeneous multimodal datasets

Metal casting process datasets can be categorized as heterogeneous and multimodal. The data is collected at different stages of the manufacturing cycle, such as process data, chemistry data, part geometry data. All of these different silos of data are individual modalities of the dataset. A lot of times, data from some of these modalities may be absent owing to different reasons. Due to missing values from individual modalities, the amount of clean data available from all the modalities for making predictions is reduced significantly. In this situation, discarding the incomplete samples that do not have all the modalities present is not helpful because the size of the dataset is reduced even further, which impacts the prediction accuracy of the machine learning models [32,33].

Furthermore, in the metal casting context, the data is missing at random, meaning that the missing values do not depend on the values of the features that are observed or present. In such a situation, if the samples with missing modalities are simply discarded, the model predictions would be highly biased. Furthermore, a whole block of features can go missing at once, meaning all features from a specific modality go missing [34,35]. For example, for some of the samples, while merging the datasets, we have data from the process data modality but do not have data from the chemistry data modality [26,36].

In such situations, it is important to impute the missing modalities with appropriate values in order to improve the prediction accuracy of the classifiers, rather than simply discard the samples with missing data in them. In order to tackle this situation, we implemented heterogeneous masked autoencoder and variational autoencoder [37, 38] techniques to impute these missing values before implementing any machine learning models for classification. Missing data imputation methods largely depend on the pattern in which the data is missing. This study shows that, as the missing rate increases, our imputation models using backpropogation on a masked matrix, prove to give better performance as compared to a naive technique such as mean imputation. The basic architecture of an autoencoder is as given below:

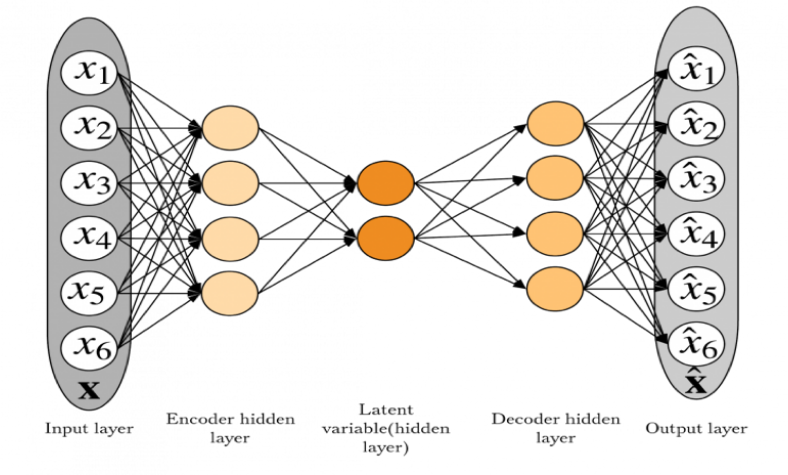


Figure 4.1: Architecture of an autoencoder [8]

Autoencoders [39] take an input $X \in R^d$, which is an input in a d-dimensional space, and map it using an encoder to a hidden representation $y \in R^k$ through a mapping, as given by the form [40]: $y = f_\theta(x) = s(W.x + b)$. Here s can be any non linear activation function such as a sigmoid or a tanh, depending on the application. In the second part of the network, also known as the decoder, y is mapped back into a reconstruction $\bar{x}$ of the input. Here, $\hat{x} = f_\theta(x) = s(W.x^T + b)$ and it follows a mapping similar to the encoding part of the network. The weights here are updated during training in a way such that the mean reconstruction error is minimized. The loss function is as shown below, which is essentially the frobenius norm of the original and the reconstructed inputs. $L(x, \hat{x}) = \min ||x - \hat{x}||^2$

The objective is to minimize this difference and if the network is successfully able to minimize this difference and reconstruct the input closely, then it has learnt a faithful representation of the input data. The encoder compresses the input to a reduced dimension whereas the decoder reconstructs the compressed input into the original dimensions. This way it is used as a dimension reduction technique to learn useful representations of the data as well as an imputation technique to reconstruct missing values in the dataset. The compressed representation of the autoencoder learns the correlations between the modalities. This allows higher order relationships in the data to be captured effectively [41]. The architecture of the autoencoder that we use in this study is as shown below:

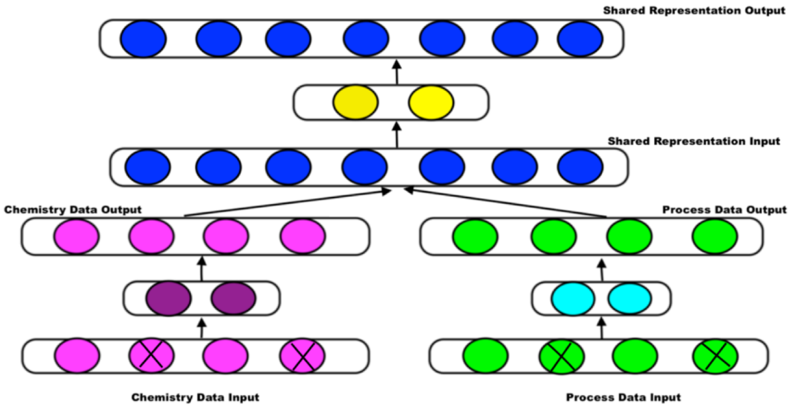


Figure 4.2: Architecture of the autoencoder used in this study.

Denoising Autoencoders(DAE) are a variation of the original autoencoder in which the input is corrupted with some form of noise, and the objective is to reconstruct the original noise-free input using a similar scheme as described above. In this thesis, we implement a form of denoising autoencoder for missing data imputation, in which the values that are missing can be considered as noise added to the original dataset and then we aim to impute these values using backpropogation with the appropriate values in a way that it minimizes the loss function. These values should be as close as possible to the original values in the dataset. This can also be thought of as reconstructing a noise free version of the input data. Denoising autoencoders, by using a different criterion of denoising, are able to extract robust features of the data, as compared to the vanilla autoencoders [41].

We implement a denoising autoencoder that is able to learn intra-modal as well as inter-modal representations of the data, in order to predict missing modalities. The multimodal denoising scheme has proven to be effective in terms of missing modality prediction as compared to using methods such as mean imputation or imputing with a value of -1. It can be seen from the results of this study that, this is a robust way of predicting missing modalities even when a significant fraction of data is missing. The Precision-Recall AUC of the imputed data almost matches that of the original dataset, even when as much as 70% of the data in the modalities is missing.

We implemented a number of different classification algorithms to test the performance. Below is a plot of the PR curve with varying missing rates using RandomForest classification. The blue line corresponds to the naive method of imputing with the mean values, the green corresponds to the imputation using our denoising autoencoder scheme, whereas the red corresponds to the original dataset. It can be seen that we are able to reconstruct the original dataset and get performance close to optimal even when a significant fraction of the data is missing.

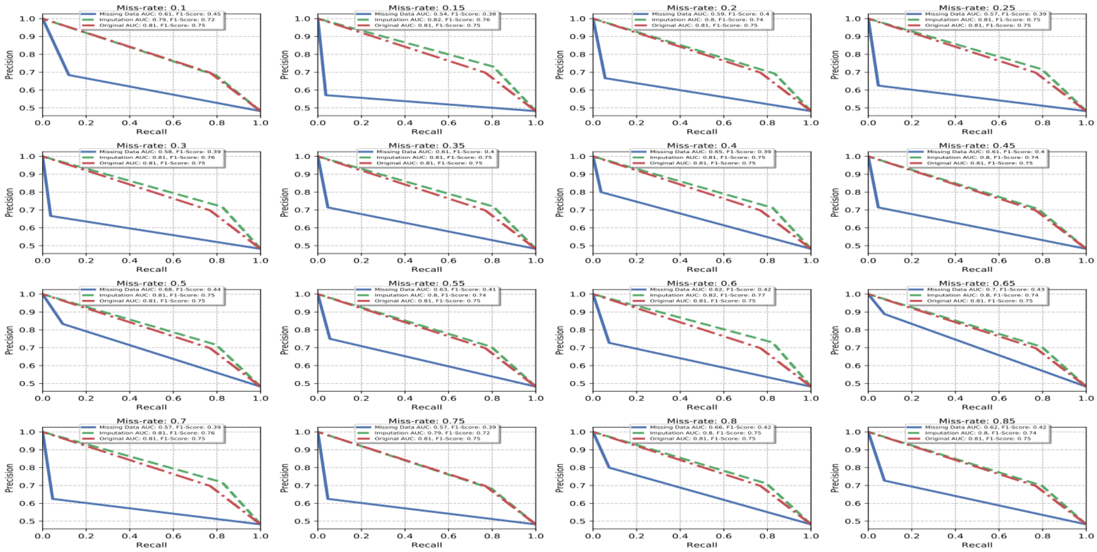


Figure 4.3: Comparison of imputation techniques with original in terms of area under the precision recall curve.

Foundries can leverage this fact and use this model to get accurate predictions on the part quality even when their system may be down or if they are not able to collect data for a particular modality, or have measurement errors during data collection. It is also very expensive to measure all the variables. Moreover, the data collection requires specific attention of the staff. Using such models will help the foundry reduce the expense by saving time and other resources.

As the results in the next few pages show, we have tested the effectiveness of this method on both the Palmer Foundry dataset as well as an open source power plant dataset obtained from UCI Machine learning repository. This dataset involves a regression problem, unlike the classification problem in the Palmer dataset. We wanted to test the effectiveness of the above method on a different scheme. This would be very useful for metal casting dataset such as the Palmer Foundry, in the future. It would be interesting to see how the models perform when we use continuous outputs such as the actual porosity values rather than only classifying the parts as pass or fail. The methodology used for implementing this masked autoencoder is as given below:

We first use only the training data to get the reconstruction error on the non-missing values. We use the trained model to obtain X-test and X-test reconstructed and use the reconstructed values for imputation. We use the reconstruction error on the non-missing values because in the real datasets, the missing values are unknown. For example, we use a masking matrix such as S^x, as given below: $S_{n,t} = 1$ if $X_{n,t}$ is observed and is equal to 0, if $X_{n,t}$ is missing. The final matrix would be $X_F = X_I.S^x$. The goal is to minimize the term below:

$$\min ||(X - X_I) \odot (1 - S^x)||^2 \tag{4.1}$$

We change the objective function to:

$$\min ||(X - \hat{X}) \odot S^x||^2 \tag{4.2}$$

Here we only include the observed values since we do not know X_I, for the modes that are missing. However, if the model does well on the data that is observed, we can safely assume that it learns the statistical relations of the features of the data and are good approximations of the truth. We can then use the same trained model to replace the actual missing values. We also want to use combinations of different modalities that are missing, and also different percentages of those combinations, to understand which once play the most important role in terms of the part qualities. Currently, the dataset that this analysis involves, has about 30% of values missing from only the process data modality. Our model can optimally recover this modality as seen in the results above. We try to emulate the amount and structure of noise such that it is similar to that found in the real dataset, in order to get suitable and relevant results. The loss function is a combination of root mean squared error as well as the KL divergence between the original and the reconstructed data. The loss function as we train across multiple epochs is seen below for the training as well as testing data until convergence:

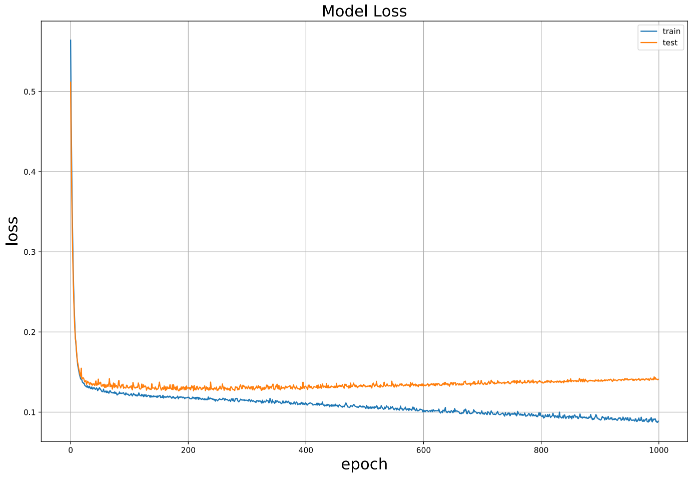


Figure 4.4: Model loss across epochs

The plot below shows the F_1score comparison between the original, reconstructed data and the data that is masked with an average value for each column, using different classifiers (missing data). We get similar results as above, however, even when imputing with -1 instead of the column average.

It can be seen that, using our multimodal denoising model, the F_1score, using any of the above classifiers almost matches the original dataset F_1score, as compared to using only mean imputation as well as imputation using -1. The above plots were generated using a miss-rate of 0.5. It can be seen that the predictions of both the classes are similar to the original. For some of the plots, the true positives are slightly higher than the original using the imputation model, however, this could only be by chance.

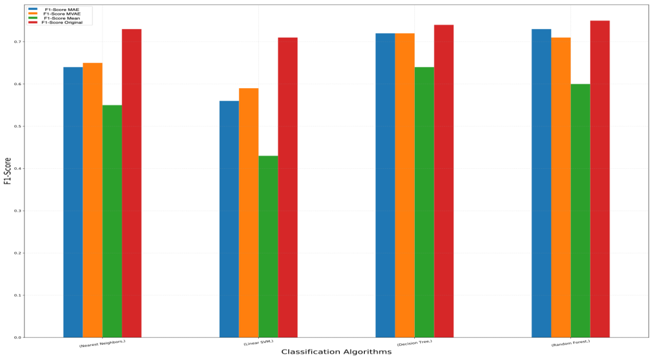


Figure 4.5: Comparison of models. The plot shows a comparison of the HMAE and HMVAE models with mean imputation and the original data. It can be seen that the HMAE and HMVAE models outperform the imputation using mean in terms of the F_1score.

We also constrained the feature space of the autoencoder to a gaussian and implemented a variational autoencoder model for the same reasons as given above. The autoencoder model makes it likely that a sampled datapoint will be from the original distribution by constraining the feature space. It learns the features that are present and also the patterns in the data during training, in order to predict the modalities that are missing.

Variational autoencoder has the architecture as shown in the image below:

Upon implementation of the above models, we observed that, the variational autoencoders slightly improved performance as compared to the original network. Variational autoencoders(VAE) are a popular deep learning approach that are being used widely in research to learn complicated data distributions. Variational autoencoders are used to get the samples from some unknown distribution and learn a model which we could sample from in a way that this model is as close as possible to the unknown distribution.

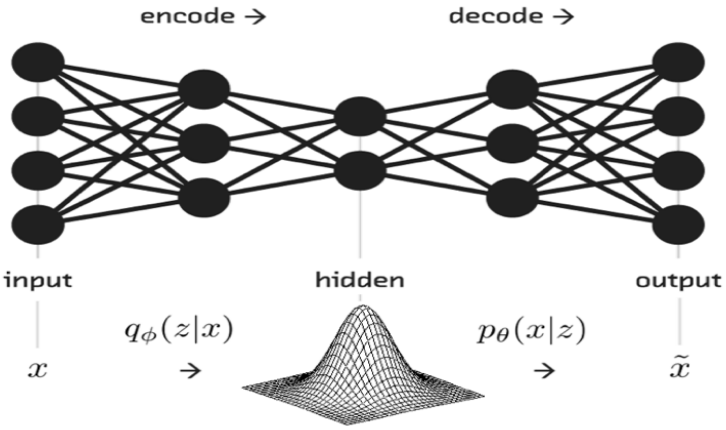


Figure 4.6: Variational Autoencoder [9]

Variational autoencoders work efficiently using backpropogation and do not make strong assumptions about the data.

The output distribution for the VAE models is typically chosen to be Gaussian. The equation above shows that it has a mean of $f(z, \theta)$ and the covariance is the identity matrix times a scalar which is a hyperparameter denoted by sigma. We can use gradient descent or other optimizations to make $f(z, \theta)$ approach X for some z. The main aim is to make the training data model very likely under the given model. The idea is to generate a distribution of k variables by taking a set of k variables that follow a normal distribution and map them through a complicated distribution.The core of variational autoencoders is explained by the equation below:

$$\log P(X) - D[Q(z|X)||P(z|X)] = \mathbb{E}_z\ Q[\log P(X|z)]D[Q(z|X)||P(z)] \tag{4.3}$$

We want to construct a Q which depends on X and makes the KL divergence between $Q(Z)$ and $P(z|X)$ small. This would make the distribution tractable. On the left hand side, we want to maximize $\log P(X)$ and minimize $D[Q(z|X)||P(z|X)]$ [42,43]. We demonstrate that using masking of missing values with the average values and backpropogation, it is possible to achieve significant improvement as compared to only mean imputation.

We use masking noise in the form of an indicator matrix to set a random fraction of the inputs to zero. We also do the same using masking with the average values for individual features and also replace with -1, similar to the methodology described earlier. The autoencoder can be trained to impute these missing values using backpropogation to update weights until the loss function reaches its minimum and converges. The loss function is once again a sum of the root mean squared error and KL Divergence. We use Root mean square error e as one of the metrics of performance, which is given as below:

$$e_x := \sqrt{\frac{||(\hat{X} - X_c) \odot (1 - S^x)||^2}{||1 - S^x||}} \tag{4.4}$$

We also use KL divergence as the other metric for the variational autoencoder model. KL divergence is given by:

$$KL(P||Q) = -\sum P \log(P)/\log(Q) \tag{4.5}$$

The loss functions for the HMVAE model are a combination of both, the RMSE as well as KL Divergence. The loss functions converge after multiple epochs of training, similar to the plot shown above.

The network was also implemented on another dataset, as mentioned above. The same can also be done on image datasets such as MNIST that are freely available online. However, for imputing pixels of image data using similar autoencoder architecture is comparatively easier to do, due to the large amount of image data being available as well as the fact that the models can leverage adjacent pixels being highly correlated.

If we were to use a supervised learning model instead, rather than an unsupervised deep learning approach, the amount of clean labelled data from all modalities available for training would be significantly small, which is a major issue for such types of heterogeneous datasets that rely on machine learning for predictions. A lot of information is deleted and lost in the remaining data and is wasted. In such an architecture, we can use labelled data with missing modalities in the next phase, in order to make predictions and we can also use unlabeled data with all modalities present for training. This way we can leverage as much information as possible to learn from the dataset while training the autoencoder. All features are first normalized in the range of -1 to 1 before implementing the network. This is done by using a simple preprocessing step like min-max scaling. Once the data is prepared, the features of a single modality are picked randomly and either replaced by either the mean of the features, or a value of -1, which we would use as a baseline. We chose the average or something like -1 instead of just replacing with 0s, in order to have a fair comparison of the methods. We first train the model on the data which has all the modalities and use that as the ground truth, noise-free input X. Then we randomly select modalities and set them to -1 or column average values. The model is then trained to produce the noise free X from noisy input. The below bar plot shows a comparison between the original and the masked autoencoder networks using different classifiers.

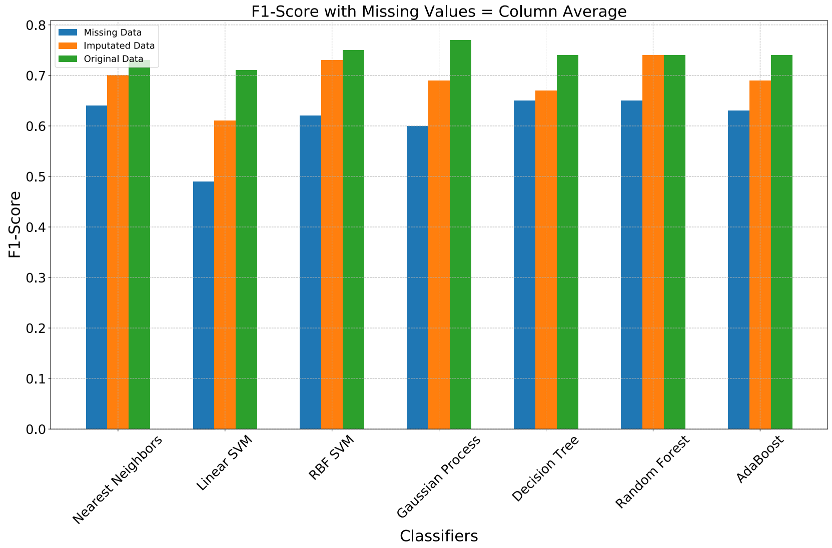


Figure 4.7: HMAE vs Mean vs Original [9]

The next plot shows the same comparison as above, except now we replace with a value of -1 instead of the mean.

In both the models, the Autoencoder performs best on all the classifiers used and the F_1score almost matches that of the original dataset. The comparison of the AE and VAE models on the power plant dataset is shown in the plot 4.9. Here we consider all the features of the dataset as a single mode. Hence, it can be considered as a unimodal representation.

The above plot shows that the performance of the models depends on which classifiers are used to calculate the F_1scores. The VAE outperforms the AE model if KNN or SVM are used as classifiers. However, both the models are still better than mean imputation.

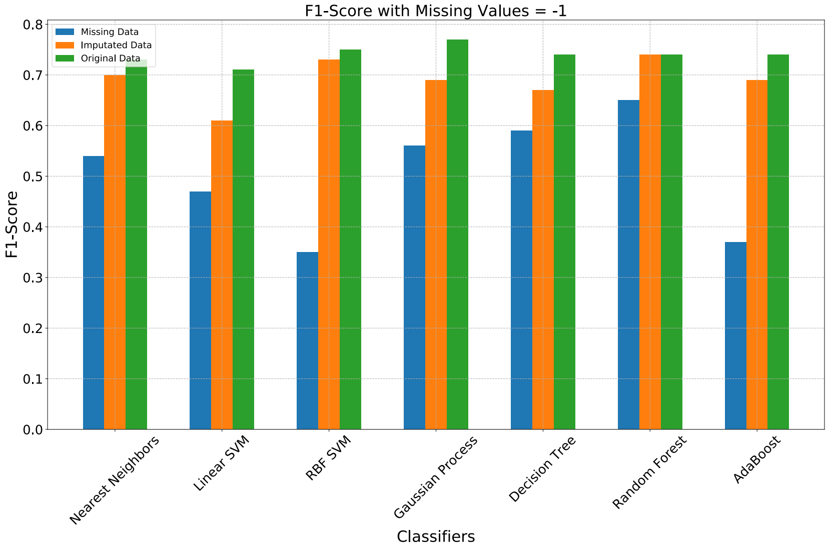


Figure 4.8: HMAE vs -1 vs Original [9]

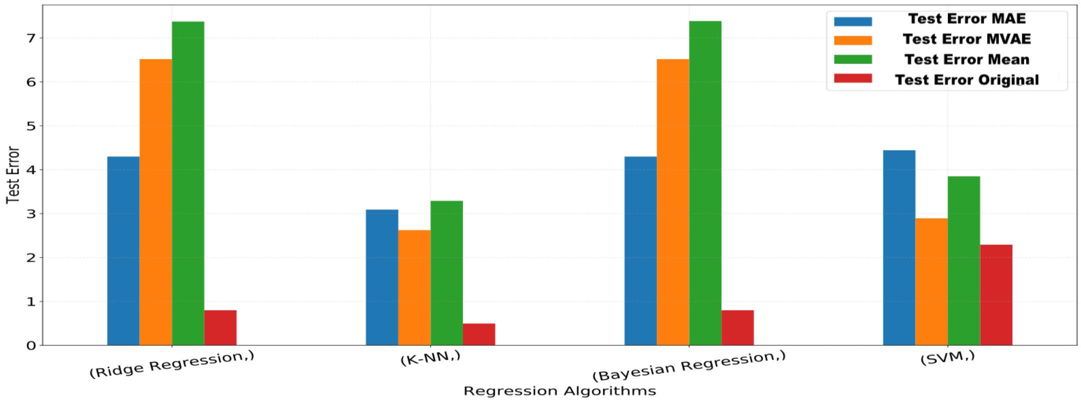


Figure 4.9: Comparison of models on regression dataset [9]

The loss function is similar to the one described above.

Finally, for the Palmer Foundry dataset, the below plot shows an example of a confusion matrix comparison between the above models using K-NN as the classifier. The diagonal elements of the HMVAE and HMAE models are closer to the original as compared to the mean imputation.

For the Palmer Foundry dataset, the above masking technique was also tried when using both the modalities of the autoencoder and reconstructing each of them indi-

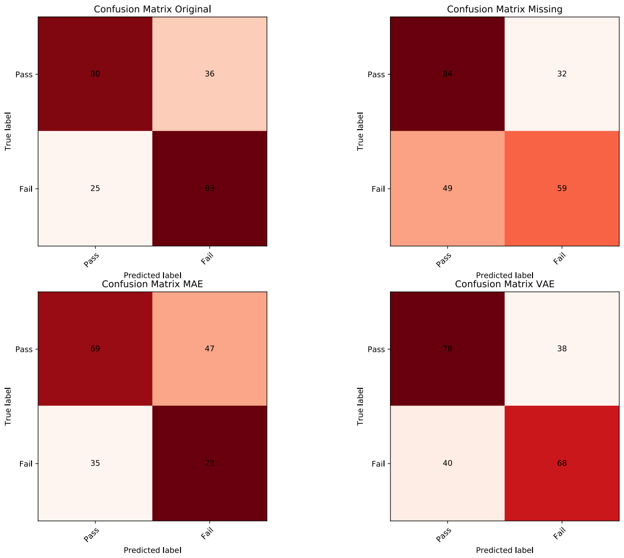


Figure 4.10: Comparison of models [9]

vidually. A separate autoencoder network is then trained on the concatenated shared reconstruction of both the modalities.

Essentially, when we use both the modalities separately and they do not share their representation, we can consider the models as Unimodal. However, at a later stage, we concatenate reconstructions of the individual modalities in order to learn inter-modal features and reconstruct them together. The models can now be considered as heterogeneous or multimodal.

The below figure 4.11 shows a comparison of the unimodal approach and it can be seen that the VAE model outperforms the AE model in terms of the F_1score, similar to the heterogeneous or multimodal case described earlier.

The F_1scores are slightly higher for both the models in the multimodal case as compared to the unimodal case. This could be due to the fact that training on individual modalities before the shared learning allows the model to learn higher order structures in the dataset.

The equations used for inter-modal and intra-modal learning using this scheme, are as given below:

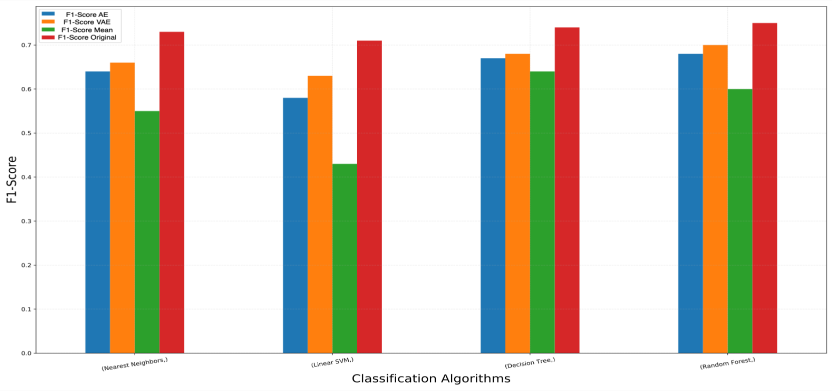


Figure 4.11: This figure shows a comparison of the F_1scores using different classifiers.

$$X_m \in \mathbb{R}^{N \times D} \tag{4.6}$$

X_m is the modality m with shape N,D.

$$S^x(n,d) = 1, if X_{n,d} \text{ is observed} \tag{4.7}$$

$$= 0, if X_{n,d} \text{ is missing} \tag{4.8}$$

$$X = X_m \odot S^x \tag{4.9}$$

$$\min ||(\hat{X} - X) \odot S^x||^2 \tag{4.10}$$

$$RMSE = \sqrt{\frac{||(\hat{X} - X) \odot S^x||^2}{||S^x||}} \tag{4.11}$$

$$J(W,b) = ||(\hat{X} - X) \odot S^x||^2 \tag{4.12}$$

$$min\ L^x(X) \ = \ ||(\hat{X} - X) \odot S^x||^2 \tag{4.13}$$

Total loss for two modalities, X and Y.

$$min\ L^{xy}(X,Y) = L^x(X) + L^y(Y) \tag{4.14}$$

The above part represents Unimodal learning or intra-modal learning of the dataset. The part below represents inter-modal learning of the dataset.

$$h \;=\; [\hat{X}, \hat{Y}] \tag{4.15}$$

where $\hat{X}$ and $\hat{Y}$ are reconstructions of modalities X and Y respectively.

$$L^h(h) = 1/2N||\hat{h} - h||^2 \tag{4.16}$$

In conclusion, using such models, we can make robust predictions on real world datasets. We can leverage the information in noisy and missing datasets. Unlike most other prediction models, our model shows that even when there is missing data in more than one modality, we can recover and improve classification performance without data loss.

Chapter 5

Conclusion

This thesis demonstrates the potential of machine learning and deep learning approaches for the quality assurance of metal casting processes. The results from this thesis show how foundries can leverage the algorithm predictions and make data driven decisions to have better control over the casting process. It will also help them save time and other resources using predictive maintenance.

5.1 Research Outcomes

- We used a semi-supervised GAN architecture which demonstrates effectiveness in synthetic data generation which is faithful to the distribution of the original data, as well as its effectiveness as a classifier. We performed in depth analysis and hyperparameter tuning and showed the limitation of a GAN by demonstrating its potential to overfit.

- We successfully implemented masked autoencoder networks for missing data imputation that almost matched the real data even when the missing rates were as high as 0.7.

- We used a multi-layer perceptron model using class weight based on ratios and validation loss, to overcome the imbalance data issue. This model was able to identify all the samples of the failed class in the unseen test data with no false positives.

- We demonstrate the value of machine learning techniques in identifying the most important features in the metal casting data sets. These features are in parallel to what the domain experts believe to be the important parameters that control this process.

5.2 Future Work

- Implement different anomaly detection techniques using deep learning, to identify the minority class samples as anomalies or outliers. In particular, we have had success using a classification scheme with classes 3,4 and 5 combined. However, we want to improve precision on each of these classes considered individually as well.
- Efforts to obtain additional data including digital X-ray images of the cast components are being carried out. The future work would include use of deep learning techniques such as convolutional neural network(CNN) for image processing to analyze casting defects.
- Work on more feature engineering and extraction, with the help of domain experts of the foundry.
- Fine tune algorithm performance on additional data and validate the models on the unseen test data.
- We would tune the losses on the GAN discriminator in order to adjust the weights based on a hyperparameter beta and compare the classification performance.

Appendix A

Machine Learning Algorithms

A.1 Machine Learning Classification Algorithms

Classification Algorithms including KNN, Random Forests and SVC were used in this study. Along with these, Ensemble learning is another approach that was tried.

Ensemble learning approach in machine learning is used because of its shown improvement in performance in terms of predictions. Ensemble learning uses the idea of using several weak learners and combining them to form a strong learner. It takes a majority vote approach in terms of classification, and this is what makes the approach more robust as compared to using single classification algorithms independently. There are different types of ensemble learning approaches, mainly Bagging and Boosting.

Bagging is a method in which multiple trees are being built over different subsets of the data.These subsets are drawn from the original dataset, with replacement. Hence, Bootstrapping is done and a model is built on each of the subsets individually. Boosting, on a high level uses algorithms that use weighted averages to convert weak learners into strong ones.

A.2 Results

The confusion matrix shows the diagonal elements as the true predictions of both the classes. The higher the number, the better the predictions, the better the classifier. The off diagonal elements are the misclassifications that are predicted incorrectly by the classifier. Confusion matrix is a good metric to use in case of unbalanced data sets, as compared to metrics such as accuracy, the reason being, shown with a simple example as given below:

If suppose in a data set, there are 99 good samples and 1 bad sample, then the classifier would be biased towards the majority class and make all predictions as belonging to the good class. If this is the case, then the accuracy of the classifier would still be 99 percent.

It is easy to tell that the classifier fails to identify the more important minority class in this case. Similarly, in the metal casting data, identifying the parts that are going to fail the quality test is more important than the majority or passed class samples. The confusion matrix makes it easy for us to see the predictions of the classifier on the minority class. The confusion matrix shows us how many samples are being identified incorrectly. They can also show us the indexes of the components that are going to fail. This will give us the percentage of correct and incorrect predictions instead of just give the raw numbers of correct and incorrect predictions for a particular data set.

It can be seen that the classifier has a hard time classifying the instances of the class 3,4 and 5 because we have such few samples of the class to learn from. In the results below, we are using a binary classification scheme, where we have all good quality parts as class 1 and all bad quality parts as class 2 , 3, 4 and 5 combined as class 2. This is why balancing the data set, before implementing any of the machine learning algorithms and assessing the performance in terms of the confusion matrix is important. The results below show the confusion matrix for the Palmer data set and the corresponding values using different algorithms after cross validation. By far, on this balanced binary scheme, the GAN is the most consistent in terms of F_1score.

A.3 Anomaly Detection

Novelty detection is another technique to deal with imbalanced data sets. The dataset is trained on one class(normal or majority class)and the algorithm forms a boundary along this class. The other class is completely ignored during the training process. Anything that is found outside of this decision boundary is classified as a novelty or an outlier. In summary, the classifier learns the patterns of the normal class and then

Table A.1: Confusion Matrices for different Algorithms.

Algorithms	**Confusion Matrix**
Random Forest	$\begin{bmatrix} 22 & 5 \\ 6 & 14 \end{bmatrix}$
Logistic Regression	$\begin{bmatrix} 22 & 5 \\ 9 & 11 \end{bmatrix}$
SVM	$\begin{bmatrix} 22 & 5 \\ 8 & 12 \end{bmatrix}$
Ensemble Learning	$\begin{bmatrix} 23 & 4 \\ 1 & 19 \end{bmatrix}$

detects the patterns that are different from the normal ones and classifies them as novel or outliers [44]. The plot showing the novelty detection performed by training on only the majority class is shown below:

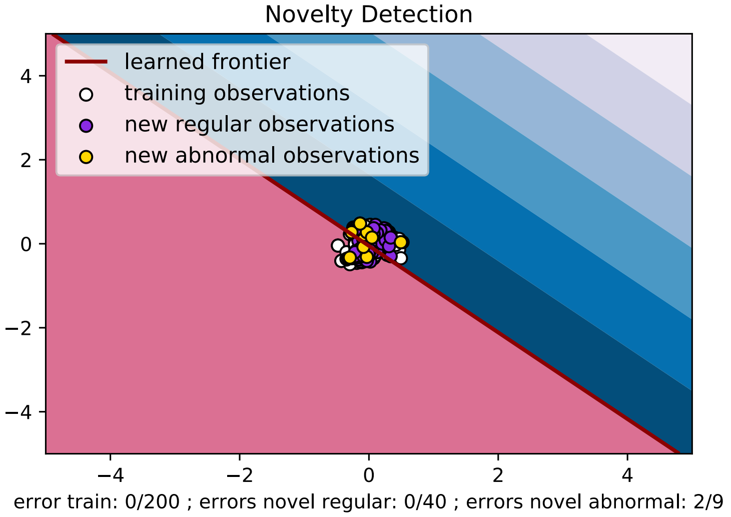


Figure A.1: Novelty Detection

As mentioned earlier, accuracy is misleading when it comes to finding metrics for classifier performance evaluation on an unbalanced data set. This is so because, even if the classifier predicts all of the instances as the majority class, even then the accuracy will be very high. However, in case of unbalanced datasets, identifying the more important minority class is the goal. This can be tested and achieved by using balancing techniques and using other performance metrics for classification evaluation, such as, precision, recall, area under the ROC curve, which will be explained in the next sections. All of these have been used in this study as part of the performance evaluation of results. The below image shows an example of a precision recall curve that is obtained using the Random Forest algorithm on the balanced data binary classification scheme.The area under this curve can be used as a single number performance of the classifier on each of the classes in the dataset. The minority or failed class in this case is defined as the positive class, whereas, the majority or passed class is the negative class.

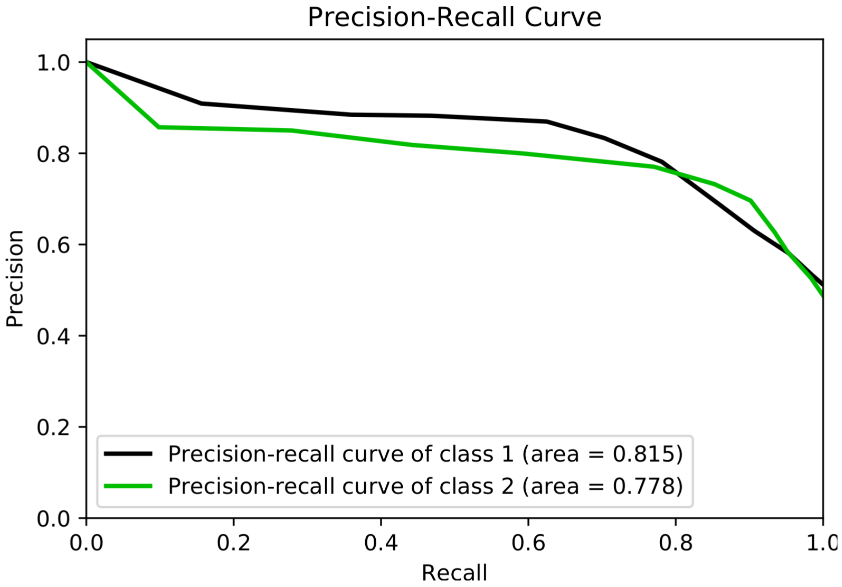


Figure A.2: Precision-Recall curve

The formulas for recall, precision and F_1score are as given below:

$$\text{Recall} = \frac{TP}{TP + FN} \tag{A.1}$$

$$\text{Precision} = \frac{TP}{TP + FP} \tag{A.2}$$

$$\text{F1Score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{A.3}$$

Bibliography

[1] N. Sun, R. Paffenroth, R. Karkare, and D. Apelian, "Machine Learning an enabling tool for advancing manufacturing competitiveness," in progress.

[2] North american die casting assocation. [Online]. Available: https://www.diecasting.org/

[3] IMPCO Inc. [Online]. Available: www.impco-inc.com/technical/

[4] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An introduction to statistical learning*. Springer, 2013, vol. 112.

[5] I. Goodfellow, "Nips 2016 tutorial: Generative adversarial networks," *arXiv preprint arXiv:1701.00160*, 2016.

[6] Log Scale. [Online]. Available: https://stackoverflow.com/questions/20384632/creating-a-log-scale-graph-in-matlab

[7] A. Odena, "Semi-supervised learning with generative adversarial networks," *arXiv preprint arXiv:1606.01583*, 2016.

[8] Autoencoder acii. [Online]. Available: https://blog.fastforwardlabs.com/2016/08/22/under-the-hood-of-the-variational-autoencoder-in.html

[9] Variational Autoencoder, VAE. [Online]. Available: https://thingsolver.com/time-series-anomaly-detection-using-a-variational-autoencoder-vae/

[10] Metalcasting Process Explained. [Online]. Available: https://www.generalkinematics.com/blog/metal-casting-process-explained/

[11] National bronze. [Online]. Available: http://www.nationalbronze.com

[12] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

[13] Principal Component Analaysis: Step by Step Explaination. [Online]. Available: https://towardsdatascience.com/a-step-by-step-explanation-of-principal-component-analysis-b836fb9c97e2/

[14] SPSS Tutorials. [Online]. Available: https://www.spss-tutorials.com/z-scores-what-and-why/

[15] Introduction to Neural Networks. [Online]. Available: https://ujjwalkarn.me/2016/08/09/quick-intro-neural-networks/

[16] Principal Component Analaysis: Three Steps. [Online]. Available: https://sebastianraschka.com/Articles/2015_pca_in_3_steps.html

[17] L. Van Der Maaten, E. Postma, and J. Van den Herik, "Dimensionality reduction: a comparative," *J Mach Learn Res*, vol. 10, pp. 66–71, 2009.

[18] K-means, Principal Component Analysis, and Autoencoder. [Online]. Available: https://web.cs.dal.ca/~kallada/stat2450/lectures/Lecture16.pdf

[19] Principal Component Analaysis: Setosa. [Online]. Available: http://setosa.io/ev/principal-component-analysis/

[20] Principal Component Analaysis: Easy Steps. [Online]. Available: https://sebastianraschka.com/Articles/2014_pca_step_by_step.html

[21] S. Mika, B. Schölkopf, A. J. Smola, K.-R. Müller, M. Scholz, and G. Rätsch, "Kernel pca and de-noising in feature spaces," in *Advances in neural information processing systems*, 1999, pp. 536–542.

[22] Principal Component Analaysis: Nature. [Online]. Available: https://www.nature.com/articles/nmeth.4346

[23] Autoencoders: Stanford UFLDL. [Online]. Available: http://ufldl.stanford.edu/tutorial/unsupervised/Autoencoders/

[24] Exploring Xgboost. [Online]. Available: https://towardsdatascience.com/exploring-xgboost-4baf9ace0cf6/

[25] R. Kohavi *et al.*, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *Ijcai*, vol. 14, no. 2. Montreal, Canada, 1995, pp. 1137–1145.

[26] 3 methods to handle missing data. [Online]. Available: https://www.datascience.com/blog/missing-data-imputation

[27] J. Benesty, J. Chen, Y. Huang, and I. Cohen, "Pearson correlation coefficient," in *Noise reduction in speech processing.* Springer, 2009, pp. 1–4.

[28] R. Blagus and L. Lusa, "Smote for high-dimensional class-imbalanced data," *BMC Bioinformatics*, vol. 14, no. 1, p. 106, Mar 2013. [Online]. Available: https://doi.org/10.1186/1471-2105-14-106

[29] GAN to WGAN. [Online]. Available: https://lilianweng.github.io/lil-log/2017/08/20/from-GAN-to-WGAN.html

[30] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, 2014, pp. 2672–2680.

[31] Gentle Intro to Generative Adversarial Networks (GANs). [Online]. Available: https://machinelearningmastery.com/what-are-generative-adversarial-networks-gans/

[32] multiple imputation. [Online]. Available: https://bmcmedresmethodol.biomedcentral.com/articles/10.1186/s12874-017-0442-1

[33] Missing data imputation: focusing on single imputation. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4716933/

[34] Multiple imputation for missing data. [Online]. Available: https://www.bmj.com/content/338/bmj.b2393/

[35] Single missing data imputation. [Online]. Available: https://bookdown.org/mwheymans/Book_MI/single-missing-data-imputation.html

[36] R. D. Camino, C. A. Hammerschmidt, and R. State, "Improving missing data imputation with deep generative models," *arXiv preprint arXiv:1902.10666*, 2019.

[37] I. Khemakhem, D. P. Kingma, and A. Hyvärinen, "Variational autoencoders and nonlinear ica: A unifying framework," *arXiv preprint arXiv:1907.04809*, 2019.

[38] Deep Learning Autoencoders. [Online]. Available: https://medium.com/datadriveninvestor/deep-learning-autoencoders-db265359943e

[39] P. Baldi, "Autoencoders, unsupervised learning, and deep architectures," in *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*, ser. Proceedings of Machine Learning Research, I. Guyon, G. Dror, V. Lemaire, G. Taylor, and D. Silver, Eds., vol. 27. Bellevue, Washington, USA: PMLR, 02 Jul 2012, pp. 37–49. [Online]. Available: http://proceedings.mlr.press/v27/baldi12a.html

[40] Understanding of Multilayer Perceptron (MLP). [Online]. Available: https://medium.com/@AI_with_Kain/understanding-of-multilayer-perceptron-mlp-8f179c4a135f

[41] Multimodal Stacked Denoising Autoencoders. [Online]. Available: https://pdfs.semanticscholar.org/b500/29d4f18ab1b7a7eeaf6c39734c4fa60e5642.pdf

[42] Tutorial on Variational Autoencoders. [Online]. Available: https://arxiv.org/pdf/1606.05908.pdf

[43] Partial Dependence Plot (pdp). [Online]. Available: https://christophm.github.io/interpretable-ml-book/pdp.html

[44] H.-j. Lee and S. Cho, "The novelty detection approach for different degrees of class imbalance," in *International conference on neural information processing*. Springer, 2006, pp. 21–30.