Pattern Classification with Missing Data Values
CSE 802: Project Group # 9

Piyush Gupta, Zhongzheng Chen, Boyang Liu, Pradyumna Reddy

May 4, 2019

1 Abstract

Missing features problem in data sets is often encountered during training stage in any pattern recognition problem. Most of the classification or regression problems handle the missing features by simply ignoring the missing values or replacing them by 0. However, such an approach is not optimal as it does not utilize any information about the data or features that might be embedded into the available dataset. Therefore, an efficient and accurate method that utilizes the available information is required to impute the missing feature values. In this project, a Mobile price classication problem is considered to study how missing data can influence our decisions to predict the price-range of the mobile based on its features such as RAM, camera resolution..etc.

2 Introduction

A person named Bob wants to start his own mobile company. He wants to give a tough fight to big companies like Apple, Samsung etc. So he has to know how to estimate the price of mobiles his company creates.To solve this problem he collects sales data of mobile phones of various companies. During the collection of data, he observed missing values for some of the mobile features which makes the mobile price estimation task more difficult.In this competitive mobile phone market we cannot simply assume things and wrong estimation of prices would have an adverse effect on his company.

So we would like to help bob by proposing some methods to deal with the missing values and to get efficient estimates for the mobile prices. So we took mobile phone data sets from https://www.kaggle.com/iabhishekofficial/mobile-price-classification. The training dataset consists of 2000 patterns,with each pattern containing 20 mobile features such as RAM, camera resolution, screen size..etc. These patterns are classified into 4 price ranges.The dataset also contains 1000 test-patterns for evaluation of our classifiers. In this project we would utilize the dataset with different percentages of missing features and try to build our classifier based on the limited information.

We would use the following evaluation metrics to judge the performance of our classifier:

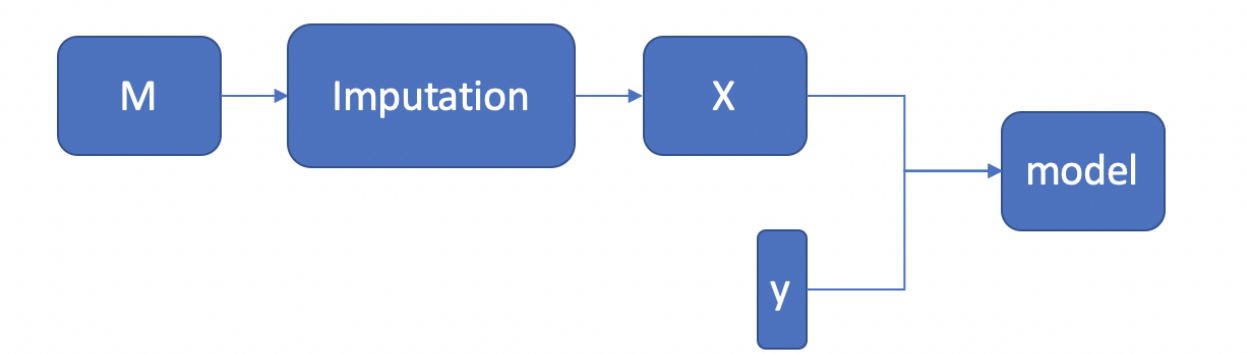

Figure 1: Pipeline of our experiment. **M** means the data with missing value, and using some imputation method to get the imputed data **X**. After imputation, we will combined with supervised information **y** to get the model

- RMSE: This metric will be used to evaluate how close our imputed data is to the real data
- AUC: This metric will be used to evaluate the final classification accuracy based on the feature with missing value

3 Problem Statement

We want to build a joint classifier capable of predicting the mobile phone price range while imputing any missing features. We want to study the matrix factorization techniques on a training set with missing features, and try to propose methods that are robust to these missing values. In particular, we would like to test the imputation methods such as mean imputation, SVD++, and Deep imputation, to study the effects of imputation on the final classification accuracy.

We are going to compare our method with different baselines:

- Mean imputation + model
- SVD imputation + model

We plan to utilize different classification models for the second-step prediction such as Bayes classification, logistic regression and Support Vector Machines. Our plan is to conduct experiments on different missing ratio of our data, and to validate how our method performs in different scenarios.

4 Technical Approach

Our method pipeline is shown in Figure 1.

4.1 Imputation Objectives

Given a matrix $\mathbf{M} \in \mathbb{R}^{n \times d}$ with missing values, where n is the number of samples and d is the feature dimension, a binary mask matrix $\mathbf{\Omega} \in \mathbb{R}^{n \times d}$ to indicate where the missing value is. Our goal is try to build a model to recover the missing data given some data with missing value. Mathematically, we want to minimize the following objectives:

$$\min_{\theta} \sum_{i} ||f(\mathbf{m}_i, \theta) - \mathbf{x}_i^*||^2, \tag{1}$$

where function f tries to impute the feature $\mathbf{x}_i^*$ is ground truth data. Clearly, if our feature $\mathbf{X}$ is a random matrix and each element is independent, then, it is impossible to impute the missing value. Also, we partial observe data $\mathbf{X}$, which makes equation 1 to be intractable. In order to impute the missing value, we have to make some assumptions. In traditional machine learning, ideally, we wish the samples are i.i.d and the features are independent to each other. However, in real world dataset, usually the data are highly correlated and features are not independent. The common assumption for matrix completion is the low rank assumption, which assume the matrix $\mathbf{X}$ is row correlated and column correlated. Then, we can transfer the problem of matrix imputation as how to fill in those missing value cells while make the whole matrix as a low rank matrix. Then, the imputation objective function is:

$$\begin{array}{ll} \text{minimize} & \text{rank}(\boldsymbol{X}) \\ \text{subject to} & X_{ij} = M_{ij}, \quad (i,j) \in \Omega \end{array} \tag{2}$$

where Ω is the set of observed data. However, this objective $\min \text{rank}(X)$ is non-convex, and it is hard to solve. In order to solve the equation 2, we need to make the convex-relaxation to make the problem easier to solve. In 2002, Fazel [1] showed that the nuclear norm $||\mathbf{X}||_*$ is the tightest convex relaxation of $\text{rank}(\mathbf{X})$. The nuclear norm is defined as follows:

$$\|Z\|_* = \sum_{j} \sigma_j(Z) \tag{3}$$

The above definition shows that the nuclear norm is the sum of singular values of matrix. Also, minimizing the nuclear norm is a convex problem with respect to the matrix itself. Thus, the equation 2 can be relaxed to:

$$\min_{X} \sum_{(i,j) \in \Omega} (X_{ij} - M_{ij})^2, \quad \text{s.t. } \|X\|_* \leq \tau \tag{4}$$

In this case, many algorithms such as singular value thresholding algorithm [2] can be used to perform the matrix completion. Singular value thresholding try to solve the above problem by using the penalty method, which introduce the hyperparameter λ, and the projection notation $\mathbf{P}$:

$$\min_{\mathbf{X}} ||P_{\Omega}(\mathbf{X}) - P_{\Omega}(\mathbf{M})||^2 + \lambda ||\mathbf{X}||_*, \tag{5}$$

where P_{Ω} denotes the projection to the observed entries:

$$P_{\Omega}(\mathbf{X}_{i,j}) = \begin{cases} 1, & \text{if } \mathbf{X}_{i,j} \text{ is observed} \\ 0, & \mathbf{X}_{i,j} \text{ is missing} \end{cases}$$

Now, the problem is a convex optimization problem, which has nice theoretical guarantee to converge to global minima. We will introduce the optimization details in next section.

4.2 Optimization for Imputation

Equation 5 is not differential objective due to the nuclear norm term. Thus, traditional methods such as gradient descent would not work here. We will use the proximal gradient descent to solve this problem. Proximal gradient descent is proposed to deal with the optimization problem in the form of differential part add non-differential part. In our case, we can rewrite the optimization problem as:

$$\begin{aligned}
&\min_{\mathbf{X}}(f(X)+g(X)) \\
&s.t. \qquad f(x) = ||P_\Omega(\mathbf{X}) - P_\Omega(\mathbf{M})||^2 \\
&g(x) = \lambda||\mathbf{X}||_*
\end{aligned}$$

Since f(z) is continuous, the gradient is

$$\nabla_x f(X) = 2(P_\Omega(\mathbf{X}) - P_\Omega(\mathbf{M}))$$

After we perform the normal gradient descent based on the differential part, we need to calculate the proximal operator for the nuclear norm.

Noting the singular value are all non-negative, and thus, the nuclear norm can be expressed as L1 norm of the singular values. So, we can directly borrow the proximal operator for lasso to solve the nuclear norm minimization problem. The proximal operator for L1-norm is also known as soft-thresholding operator. The soft thresholding operator $S_\lambda(\beta)$ is defined as follows:

$$[S_\lambda(\beta)]_i = \begin{cases} \beta_i - \lambda & \text{if } \beta_i > \lambda \\ 0 & \text{if } -\lambda \leq \beta_i \leq \lambda \\ \beta_i + \lambda & \text{if } \beta_i < -\lambda \end{cases} \tag{6}$$

After we get the proximal operator, we can use the common proximal gradient descent algorithm to solve the optimization in equation 5 as follows:

$$\begin{aligned}
&X^{(k+\frac{1}{2})} = X^{(k)} - \eta\nabla_x f(X^{(k)}) \\
&\text{perform SVD on } X^{(k+\frac{1}{2})} \\
&X^{(k+\frac{1}{2})} = U\text{diag}(\sigma)V^T \\
&\sigma' = S_\lambda(\sigma) \\
&X^{(k+1)} = U\text{diag}(\sigma')V^T,
\end{aligned} \tag{7}$$

where η is the learning rate. The imputation algorithm can be summarized as follows:

Algorithm 1 Imputation Algorithm

Input: Data matrix $\mathbf{M}$, Missing indicator Ω, hyperparameter λ, learning rate η, Stopping Criteria ϵ
Output: Imputed matrix $\mathbf{X}$
begin
  random initialize $\mathbf{X}$ as $\mathbf{X}^{(0)}$, and f_{old} as a very large number
  for i = 1:maxiter
    Using Eq. 7 to update X
    Evaluate loss using Eq. 5 to get f_{new}
    if $(f_{new} - f_{old})^2 \leq \epsilon$
      break
    $f_{old} = f_{new}$
  end for
end

4.3 Build Classifier Based on Imputed Data

After we imputing the data matrix, we can build different classifiers using imputed data. In this project, we use three types of classifier, which are Naive Bayes Classifier, Logistic Regression, and Support Vector Machines.

4.3.1 Naive Bayes Classifier

Given a dataset, we are only able to compute the class specific prior probability ω_c and the likelihood $p(\boldsymbol{x}|\omega_c)$. Therefore, we can use the Bayse Rule to compute the posterior probability $p(\omega_c|\boldsymbol{x})$.

$$p(\omega_c|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|\omega_c)P(\omega_c)}{P(\boldsymbol{x})} \tag{8}$$

We decide to implement a very basic Naive Bayes Classifier where we are going to make the assumption for 0-1 loss functions. In this case, the classifier will predict the class for unknown test samples based on searching for which class gives the point the maximum posterior probability.

$$c^* = \text{argmax}_{c \in C}(-R(\alpha_j|\boldsymbol{x})) \tag{9}$$

$$c^* = \text{argmax}_{c \in C}\left(-\sum_{i \neq c} \lambda(\alpha_i|\omega_c)p(\omega_c|\boldsymbol{x})\right) \tag{10}$$

$$c^* = \text{argmax}_{c \in C}\left(p(\omega_c|\boldsymbol{x})\right) \tag{11}$$

where $\lambda(\alpha_j|\omega_c)$ is the loss function in which we are using 0-1 loss.

There are two ways for use to estimate the likelihood probability $p(\boldsymbol{x}|\omega_c)$, either parametricly or non-parametricly. We decided to use the parametric techniques that we've learned in class to make our estimations. As what was discussed in class that assumption of gaussianality is typically reasonable for classification, we fit the data with a Gaussian distribution using maximum likelihood estimation.

$$\hat{\boldsymbol{\mu}}_c = \frac{1}{n_c}\sum_{k=1}^{n_c} \boldsymbol{x}_{c,k} \tag{12}$$

$$\hat{\boldsymbol{\Sigma}}_c = \frac{1}{n_c}\sum_{k=1}^{n_c} (\boldsymbol{x}_{c,k} - \hat{\boldsymbol{\mu}}_c)(\boldsymbol{x}_{c,k} - \hat{\boldsymbol{\mu}}_c)^t \tag{13}$$

where n_c is the number of training samples in class c.

4.3.2 Logistic Regression

Logistic Regression is a common statistical model often used to make a binary decision ('yes' vs 'no', 'even' vs 'odd') using a logistic function. It is a regression analysis which is used when the dependent variable is dichotomous (binary). Since we are dealing with a multiple class problem, we use the one vs all approach in which a logistic regression model is learned for each class. Let $c \in \mathbb{Z}_{>0}$ be the total number of classes. In one vs all approach, while modeling class i, $i \leq c$, all the samples from class i are considered with label 1 and all other samples with label 0. Let $x^{(i)}$ be a training pattern with normalized features and $h_\theta(x^{(i)})$ be the hypothesis for the logistic regression. We use a linear perceptron model with a sigmoid function as the hypothesis for each class, i.e., $h_\theta(x^{(i)}) = g(\theta^T x^{(i)})$, where θ is the vector of unknown weights to be learned for each class and $g(z) = \frac{1}{1+e^{-z}}$ is the sigmoid function. The first element of θ vector, i.e. θ_0 corresponds to the bias term.

To learn the weights of each model we use the following cost function $J(\theta)$.

$$J(\theta) = \frac{1}{n}\sum_{i=1}^{n}[-y^{(i)}log(h_\theta(x^{(i)}) - (1-y^{(i)})log(1-h_\theta(x^{(i)})] + \frac{\lambda}{2n}\sum_{j=1}^{d}\theta_j^2, \tag{14}$$

where n is the number of training samples, d is the dimension of the feature vector $x^{(i)}$, and λ is the regularization parameter to keep weights small. We use the optimized version of the fmincg solver to learn the weights for each model which utilizes the cost function $J(\theta)$ and its gradient. The gradient of cost function with respect to weight vector θ is given by:

$$\frac{\partial J(\theta)}{\partial \theta_0} = \frac{1}{n}\sum_{i=1}^{n}(h_\theta(x^{(i)}) - y^{(i)})x^{(i)}, \tag{15}$$

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{n}\sum_{i=1}^{n}(h_\theta(x^{(i)}) - y^{(i)})x^{(i)} + \frac{\lambda}{n}\theta_j, \tag{16}$$

Using the cost function and its gradient the solver finds the weights which minimizes the cost function. Finally, the predicted class for any pattern x, is given by:

$$c^* = \text{argmax}_{c\in C}(\theta_c^T x), \tag{17}$$

where θ_c is the learned vector of weights for modelling class c.

4.3.3 Support Vector Machines

In a Support Vector Machines (SVM), instead of modelling each class, a decision boundary is learned to separate the classes. The boundary is assumed to be linear in a transformed space which is obtained by using a kernal function. The linear decision boundary uses an input from a non-linear basis/kernal function which operates on the original training pattern and transforms the feature space. For our SVM classifier we use a gaussian kernal function and use standard matlab libraries for the implementation. The SVM decision boundary for any binary classification is obtained by minimizing the following cost function.

$$\min_{\theta} \lambda[\sum_{i=1}^{n} y^{(i)} cost_1(\theta^T f^{(i)}) + (1 - y^{(i)}) cost_0(\theta^T f^{(i)})] + \frac{1}{2} \sum_{j=1}^{d} \theta_j^2 \tag{18}$$

where $f^{(i)}$ is the similarity/kernal function which is chosen as gaussian in our case. The $cost_0$ and $cost_1$ are the costs corresponding class label is 0 and 1, respectively. We again utilize the one vs all approach to separate all classes.

5 Experiment Results

5.1 Synthetic Experiment

In Synthetic Experiment, we generate the low rank data matrix to see whether our method can imputed the matrix correctly. We also would like to figure out how our method perform under different missing ratio.

The synthetic data generation steps are as follows:

- generate 100*10 matrix with rank equals to 5
- generate binary mask to indicate the place of missing values
- impute the data matrix
- calculate the RMSE

We conduct the experiment in three different missing ratio settings. We run the algorithm when missing ratio is 0.3, 0.5, and 0.7.

The results are in table 5.1. The visualization of the imputation is in Figure 2, 3 and 4

In the figure, we can see that our method always converges due to the convex relaxation. Overall, with the increasing of missing ratio, the rmse is increasingly large.

missing ratio	RMSE
0.3	1.3391
0.5	2.2301
0.7	2.8776

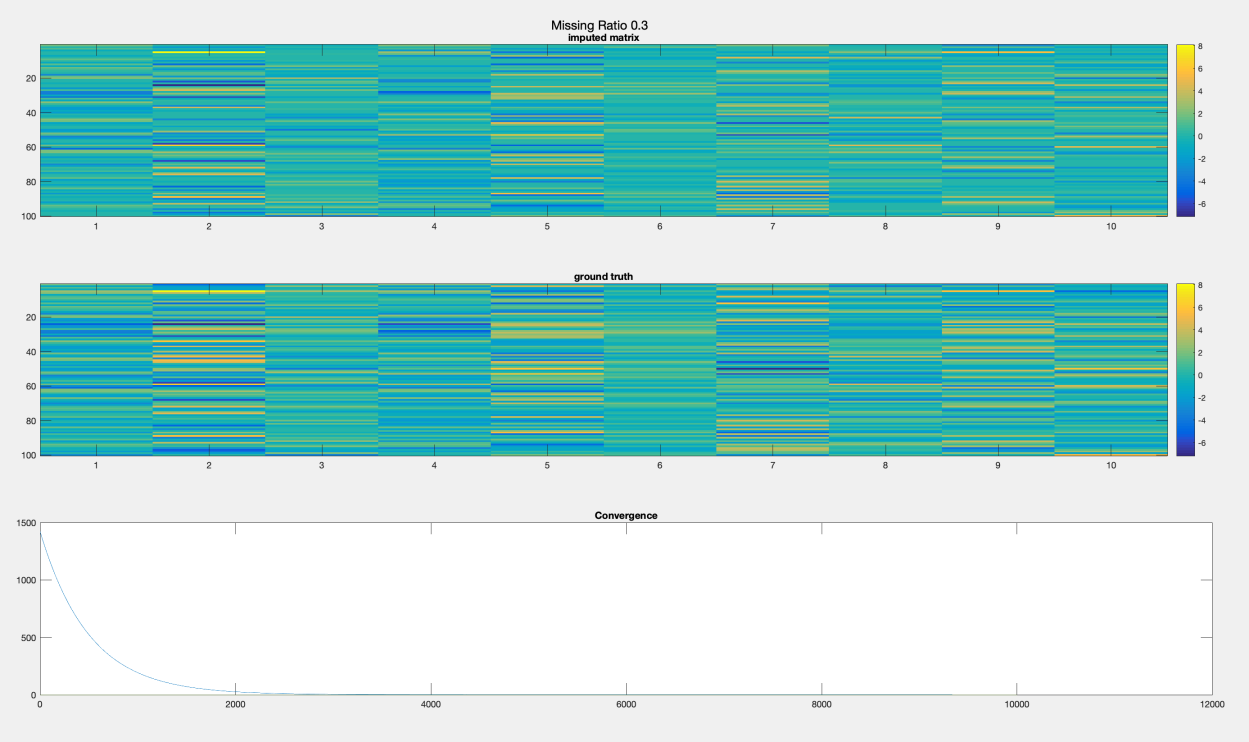

Figure 2: Imputation Results when missing ratio is 0.3, the top matrix is imputed matrix, the second row is ground truth, and the last curve is the convergence curve

5.2 Imputation with Classifiers

We conducted performance analysis of the imputation methods by observing how well the classifiers learned from the imputed data with a variety of missing rates in comparison to the original data.

Here was how we setup our experiments:

1. We generated a masking matrix with the missing rate to simulate the missing value problem in real life.
2. We use the imputation methods, either mean imputation or our proposed imputation method to impute the missing values back into the data.
3. We train the classifier with the imputed data and obtain the weights.
4. We use the trained weight to make predictions with the original data.
5. We compute the accuracy as the performance for the learned classifier.

The results of the performance of the Naive Bayes Classifier with Gaussian estimation using mean imputation are shown in Figure 5 and the performance with the proposed method are shown in Figure 6.

The results of the performance of the Logistic Regression model are shown in Figure 7.

The results of the performance of the SVM model are shown in Figure 8.

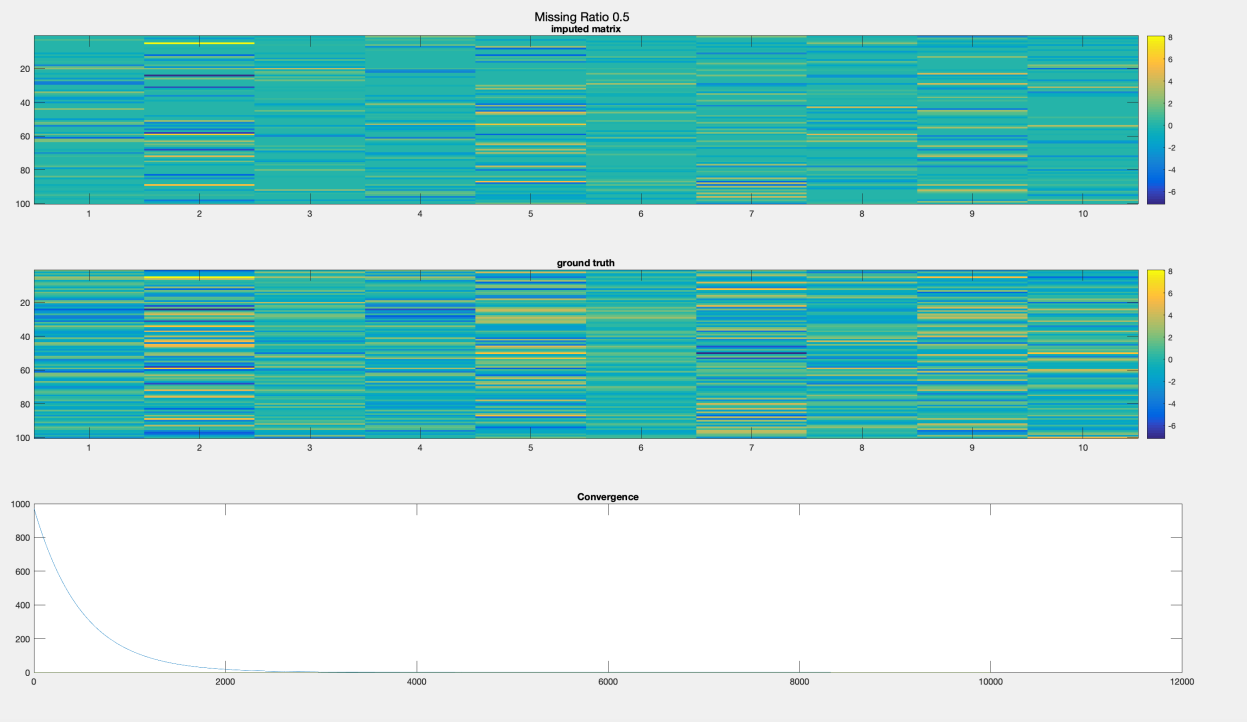

Figure 3: Imputation Results when missing ratio is 0.5, the top matrix is imputed matrix, the second row is ground truth, and the last curve is the convergence curve

Unfortunately, due to the time limitations of our project, we weren't able to obtain the performance results of Logistic Regression model and SVM with the proposed method with all the missing rates we had for the other analysis. However, we've obtained their performance with 30% and 50% missing rate, because these were where the performance differed significantly from Naive Bayes Classifier to Logistic Regression model and SVM. The accuracy for Logistic Regression was 69% and 64% and the accuracy for SVM was 34.85% and 28.57% at 30% and 50% missing rate.

6 Findings

In general, we observe that the accuracy decreases as we increase the missing percentage of features for test datasets using mean imputation for all the classification models. We could observe that accuracy decreases more linearly when bayes classifier is used along with the mean imputation. Also, the accuracy decreases until missing percentage of 70 and then increases for missing percentage of 80 in the case of mean imputation with logistic regression. In the case of SVM classifier with mean imputation we could observe that the accuracy becomes constant after a missing percentage of 70.

So we could conclude that the gaussian classifier along with both mean and SVD imputation would effectively help Bob in classification of his mobiles to the true price ranges as for most of the missing percentages it has greater accuracy than the other classifiers and also has more linear response than the others.

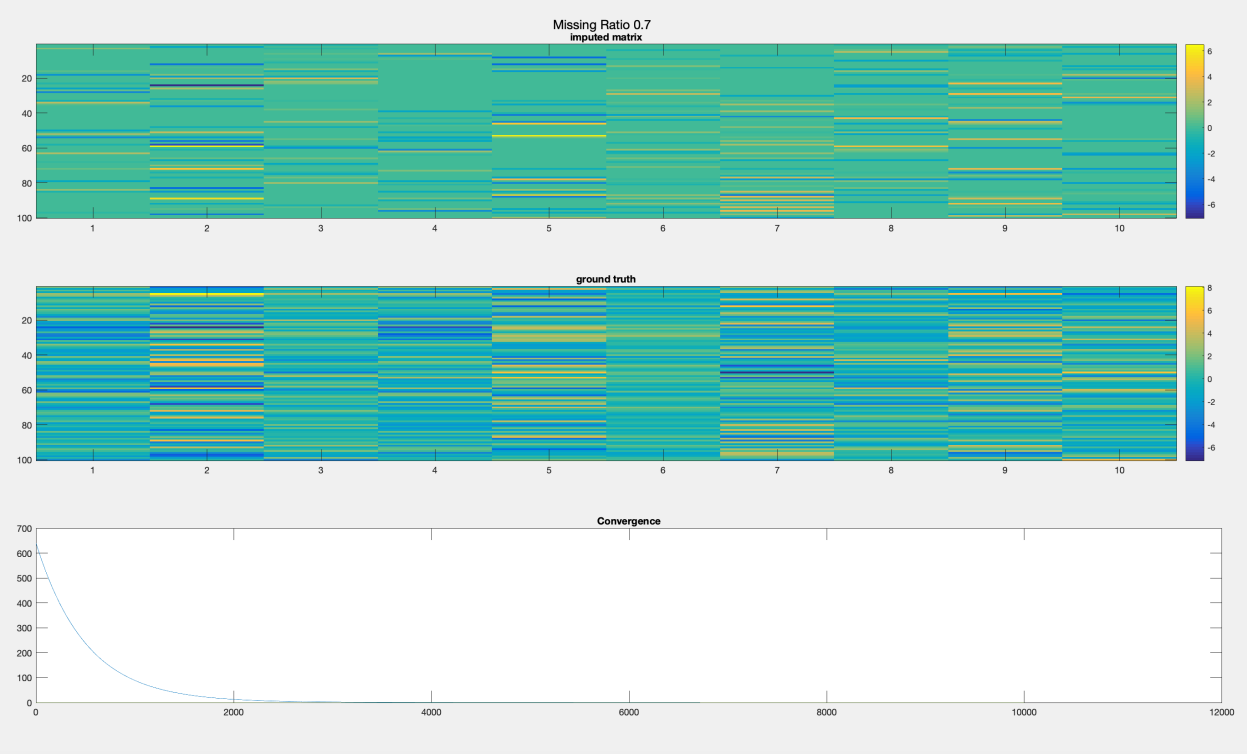

Figure 4: Imputation Results when missing ratio is 0.7, the top matrix is imputed matrix, the second row is ground truth, and the last curve is the convergence curve

7 Summary and Future Work

We have made an efficient model for Bob to correctly classify his mobiles to the price ranges even though there are missing features in the data collection. We have used mean SVD imputation models, combined with the gaussian, logistic regression and SVM classifiers to comeup with 3 overall classification models and have studied them. As the part of future work we would like to use SVD Imputation models along with logistic regression and SVM classifiers, and also other imputation models to come up with more classification models to give more insights to Bob and help him with the mobile price classification.

We would like to thank Dr.Arun Ross for helping us to learn the classification models in the course and would like to thank ourselves for working as a team and contributing equally for this successful study.

8 Bibliography

References

[1] Maryam Fazel. *Matrix rank minimization with applications*. PhD thesis, PhD thesis, Stanford University, 2002.

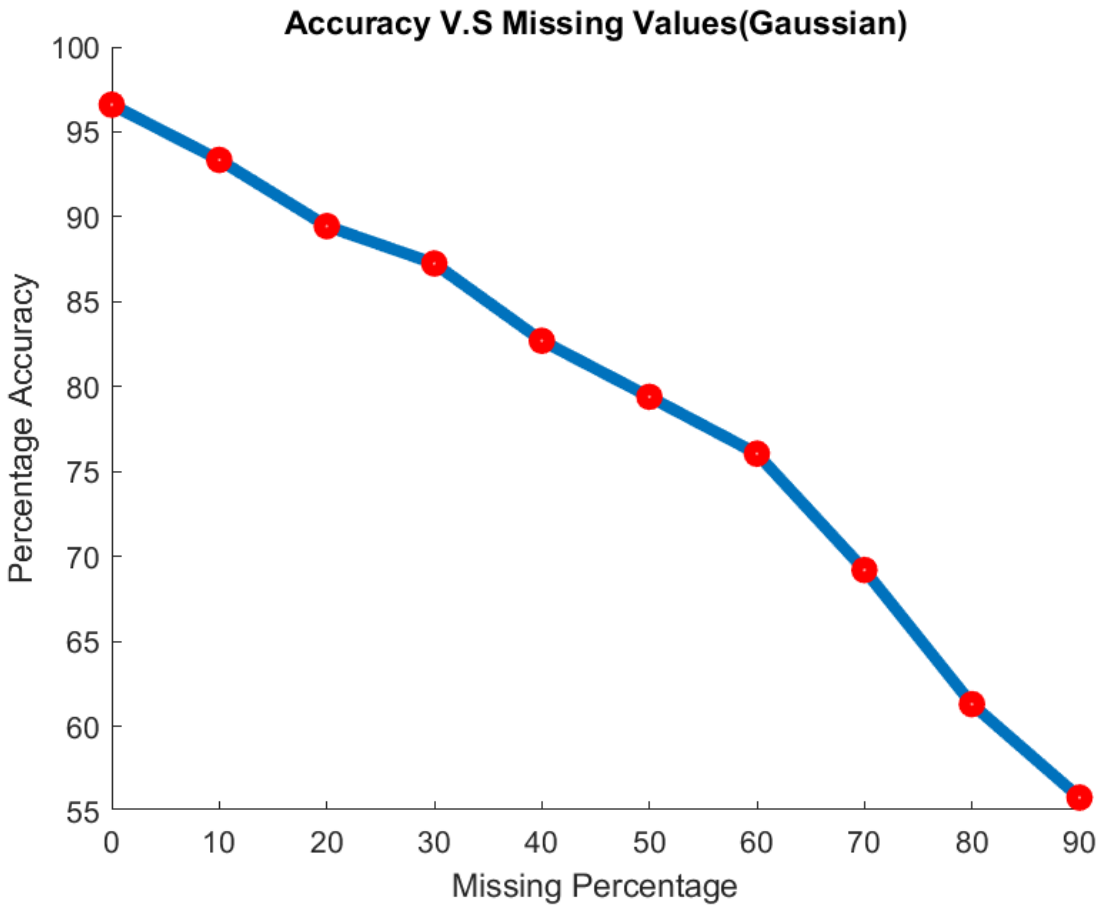

Figure 5: The performance of the Naive Bayes classifier with increasing percentage of missing features. We choose a multi-variate Gaussian as the form of the likelihood distributions. The classifier is trained with the mean imputed features and performance is evaluated on the original data set without missing values.

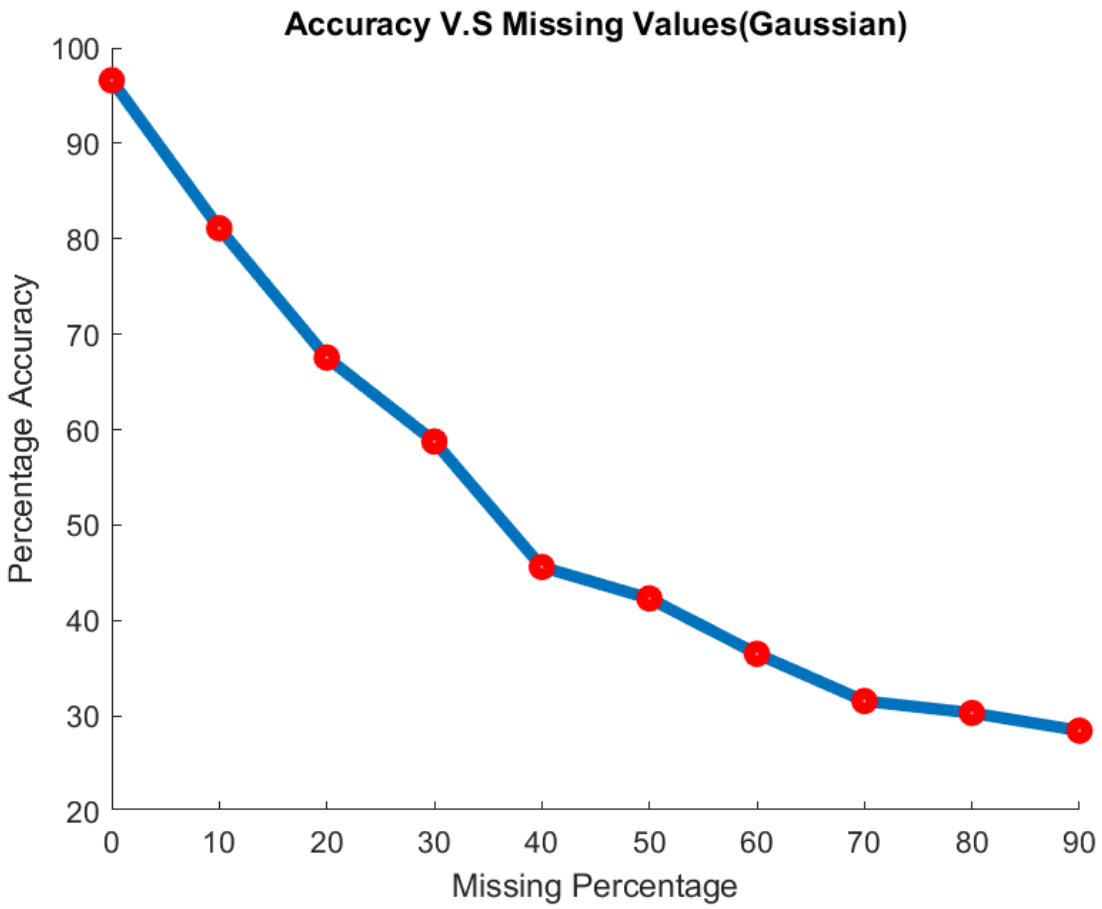

Figure 6: The performance of the Naive Bayes classifier with increasing percentage of missing features. The classifier is trained with the imputed features with our proposed imputation method and performance is evaluated on the original data set without missing values.

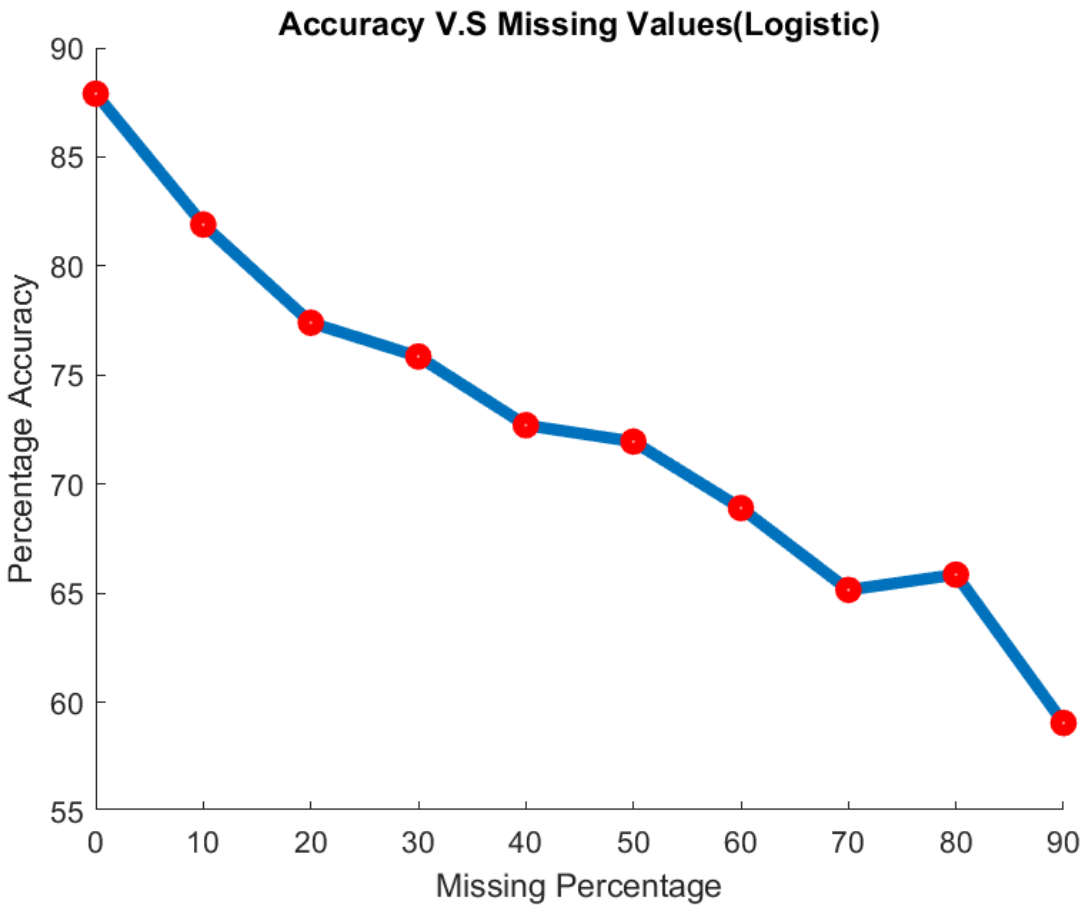

Figure 7: The performance of the logistic regression classifier with increasing percentage of missing features. The classifier is trained with the mean imputed features and performance is evaluated on the original data set without missing values.

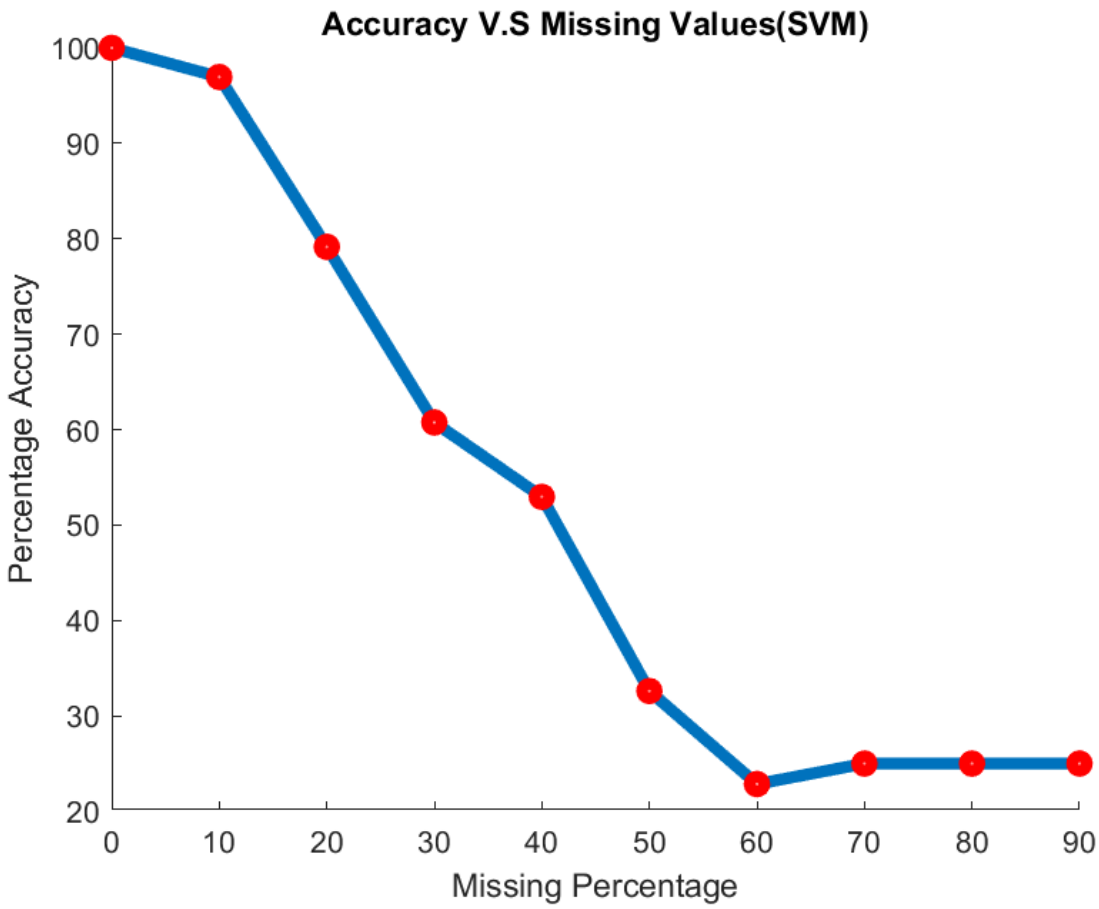

Figure 8: The performance of the support vector machine (SVM) classifier with increasing percentage of missing features. The classifier is trained with the mean imputed features and performance is evaluated on the original data set without missing values.

[2] Jian-Feng Cai, Emmanuel J Candès, and Zuowei Shen. A singular value thresholding algorithm for matrix completion. *SIAM Journal on Optimization*, 20(4):1956–1982, 2010.