CSE 506 - Data Mining Project Report: Insurance Risk Assessment

Katari Saketh
2015045
katari15045@iiitd.ac.in

Saurabh Kumar
2015089
saurabh15089@iiitd.ac.in

Shyam Agrawal
2015099
shyam15099@iiitd.ac.in

1 Introduction

To get a life insurance a person has to take medical exams, provide extensive information about themselves, etc. to identify the risk level and eligibility of their application. Evaluating risk is a long and tedious process which can take on average 30 days[1]. People don't like such a long process, and as a result, only 40% of the population takes life insurance[1].
Therefore, if we could make this process quicker and less labor-intensive, then it would be easier for people to get life insurance. The goal is to develop a model which can predict how risky it is to give life insurance to a particular person.

2 Problem Statement

Given a life insurance application and the applicant's details like medical history, employment history, age, etc., determine how risky it is for an insurance company to accept or approve the insurance application, on a scale of 1 to 8. It is this kaggle competition.

3 Dataset

This dataset is from the kaggle competition on the same problem[2].

- 59.4k Train Samples
- 19.8k Test Samples
- 127 Features
- 8 Class Labels

3.1 Description of features

The dataset has 127 features and has missing values as well. A brief description of the features of this dataset is given below:

- **Product Info:** The product applied for.
- Age of applicant.
- Height of applicant.
- Weight of applicant.
- BMI of applicant.
- The employment history of the applicant.
- The family history of the applicant.
- The medical history of the applicant.
- The presence/absence of a medical keyword being associated with the application.
- **Class Labels** (Only present in training data)

3.2 Types of features

We have the following types of features in our dataset:

- There are 61 categorical features, and none of them have missing values.
- There are 13 continuous features with missing values.
- There are 53 discrete features with missing values.

3.3 Data Visualization

We reduced the dimension (number of features) of the dataset to 2 using two different dimensionality reduction techniques namely PCA[3] and T-SNE[4]. Then we plotted 2-dimensional scatter plot giving each class a different color. The data visualized using PCA and T-SNE is shown in Figure-2 and Figure-3 respectively. As we can see from these two plots, the data is clearly not linearly separable.
Also, the distribution of data points among the 8 class is not uniform as shown in figure 1.

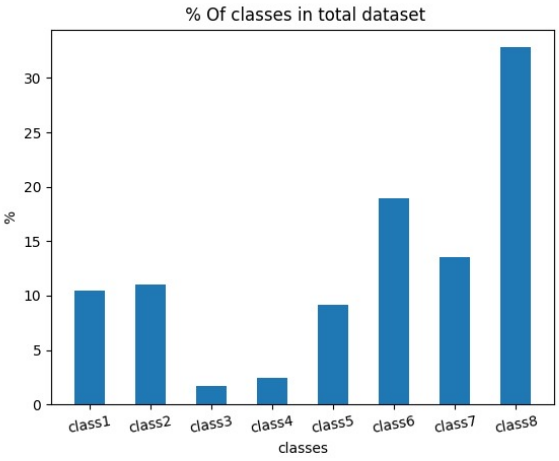

Figure 1: % of a class in total dataset

3.3.1 Why two different techniques?

Why do we need 2 visualization techniques, instead of just 1, either PCA or T-SNE?
It is because PCA just gives you the direction(s) or the dimension(s) along which, more data are present, it doesn't preserve the structural information of data. On the other hand, T-SNE preserves the structural information of data along the direction(s), where there is more concentration of data.

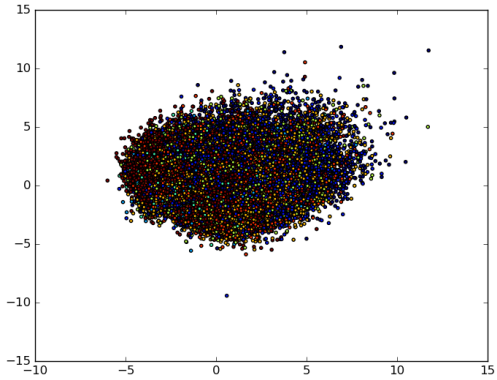

Figure 2: PCA Visualization

4 Preprocessing of data

The dataset has different types of features like categorical, continuous, etc. Some features have string values while some have integer and float values as well. Also, there are missing values in the dataset.

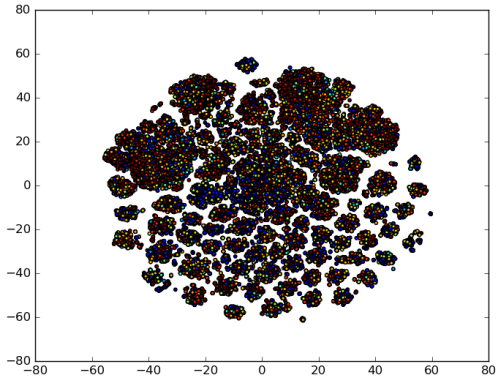

Figure 3: T-SNE Visualization

4.1 Handling String Values

Why don't we pass string data directly to an ML model like neural networks etc? Machine learning models need numerical data as input, so that, they can do some mathematical optimizations, like, minimizing the value of cost function using gradient descent etc.

The dataset has a feature (`Product_Info_2`) that has string values in it; these are converted to numerical values; for each character in the string, extract the ASCII value of the character; this ASCII value is appended (without adding them) to the previous ASCII values of the same string. For example, *ab* is converted to *9798*.

4.1.1 Why not use unique integers instead?

Why don't we just assign unique integers starting from 0, for every new string we encounter? With this approach, let's say, while training, if you encounter the string *ab* first, *0* is assigned to it; for the next string *cd*, *1* gets assigned. Now while testing, if you get *cd* first, *0* would be assigned to it; note that the same string, *cd*, has been mapped to *1* during training; from classifier's perspective, 1(*cd*) during training is different from 0 (*cd*) during testing, which, is not intended.

In continuation to the unique integers approach in the previous paragraph, why don't we store the mapping of strings and integers, and use that mapping to retrieve the integers from strings, while testing? This would solve the issue, but, if testing is not done just after training, then you would need to store the mapping on a disk (so that you could retrieve the mapping, while testing), which, is time consuming (Disk I/O is costly).

Note that, you could also use one-hot encod-

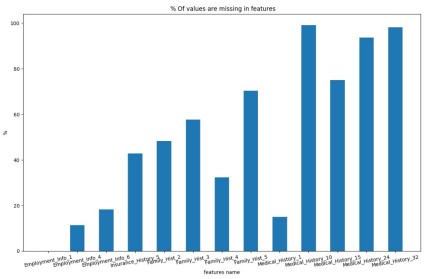

Figure 4: % of a feature that are missing

ing to convert string data to numerical data; the ASCII value approach might not be good for large strings, because, it might generate huge numbers that might exceed the limit if *int* or *long*. In our data set, we have strings of size 2, which, doesn't cause any overflow.

4.2 Handling Missing values

By default, the missing values might be filled with some garbage values like, *NaN (Not a Number)*, in case of *pandas* in Python. Doing computations on top of *NaN* would yield an other *NaN*, which is not intended.

One approach to handle missing values could be, ignore the entire sample or data point, if at least one of its feature values is missing. This is not reliable, because, you might get a test sample, which has one or more missing values in it, and you simply can't ignore/stop predicting a test sample.

In our dataset, the categorical features don't have any missing values; the numerical features, have some missing values. The amount of missing values in a feature varies from one feature to another. We have plotted the percentage of a numerical feature that are missing as shown in figure-4.

The missing values of numerical features are being replaced with the mean of the respective feature values.

An alternative way of dealing with missing values of numerical features would be to transform the distribution of values of that feature to make it more normal distribution-like (this can be done using BoxCox transformation[6][7]) and then simply put zeros in place of missing values.

If we had any missing values in categorical features, then, we would have replaced those missing values with the median of the feature values.

4.3 Normalization of Data

Machine learning models perform better, when, the data lies with in a small range, rather than being distributed non-uniformly over a very large range. You would observe an increase in accuracy, when you feed a neural network with normalized data. How? Consider Sigmoid as an activation function in a neural network; for extreme values (values that are far away from zero), the gradient (slope) would always be close to zero, which makes gradient descent ineffective and would make neural network to form an inefficient decision boundary (activation function adds non-linearity to the decision boundary).

Why normalization? Why not Min-Max Scaling? Min-Max scaling is sensitive to outliers - an outlier can act as a minimum or maximum value and can dominate the output of min-max scaling. On the other hand, normalization uses statistical metrics like mean, and *normalizes with standard deviation*, which, makes small no. of outliers almost ineffective in deciding the output of the normalized value.

Normalization also gives the data in a normal distribution, zero mean, and unit variance. It would be easier for machine learning models to perform some mathematical optimizations on this kind of data. (log of normal distribution gives a quadratic equation, which can be conveniently used in the algorithms like gradient descent)

4.4 Feature Engineering

4.4.1 Correlation Analysis [5]

Why do we need this? Consider a case where 2 features are conveying you almost the same information; you could simply ignore one of the features, without losing much information.

But, how would you know if two or more features are conveying the same information or exactly opposite information? Correlation coefficient tells you how strongly 2 variables are correlated; it always lies between -1 and +1; values in range 0.7 to 1 and -0.7 to -1 can be considered as highly correlated i.e conveying almost the same or opposite information respectively.

Why not Co-variance? Co-variance tells whether two variables increase together i.e. positive co-variance, or if one increases, the other decreases i.e. negative co-variance; it doesn't say about the scale or how strong these two variables are related to each other; the strength of the rela-

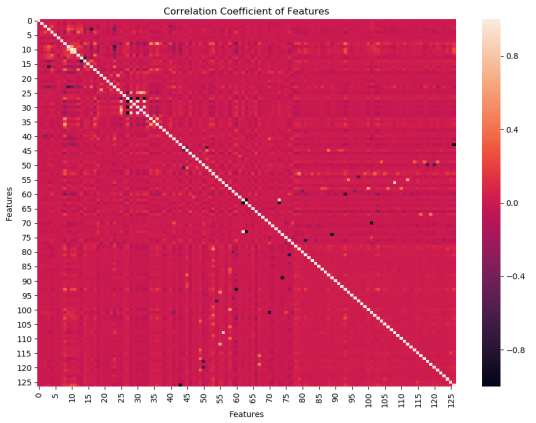

Figure 5: Correlation Coefficient of all the features with respect to every other feature; each cell being the correlation coefficient between 2 features; notice that the histogram is symmetric i.e for any two features `feature_a` and `feature_b`, correlation coefficient between `feature_a` and `feature_b` is same as correlation coefficient between `feature_b` and `feature_a`; also, the diagonal represents the value 1 (white in color) i.e correlation coefficient of a feature with itself is 1.

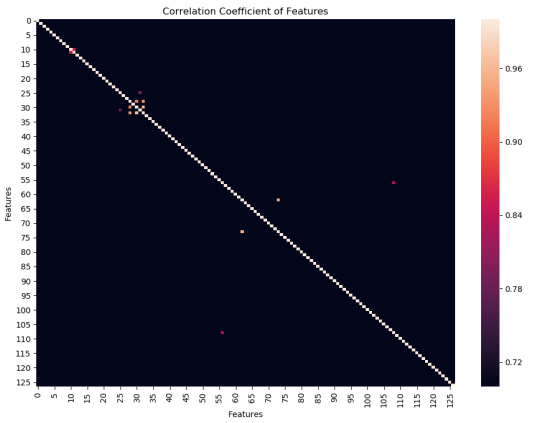

Figure 6: Highly positively correlated feature pairs (correlation coefficient between 0.7 and 1) can be seen as non-blue dots

Figure 7: Highly negatively correlated feature pairs (correlation coefficient between -0.7 and -1) can be seen as non-pink dots

tionship between two variables can be determined by correlation coefficient.

For every feature, correlation coefficient is calculated with every other feature, which generates a two-dimensional matrix, each cell value representing the correlation coefficient between two variables; the same can be seen in Figure-5.

Based on the values of correlation coefficient, we have identified 16 highly correlated feature pairs (overlapping features in feature pairs are counted only once); 7 pairs of features are highly positively correlated, whose correlation coefficient lies between 0.7 and 1, as shown in figure 6.

13 pairs of features are highly negatively correlated, whose correlation coefficient lies between -0.7 and -1, as shown in figure 7.

Most of the data points are uncorrelated i.e correlation coefficient is close to zero, as shown in figure 8.

4.5 Feature Importance with Extra Trees [8]

Why do we need to use Extra Trees to get the feature Importance? Extra trees, also known as, Extremely Random Trees, add one more step of randomness on top of Random Forests, by randomly selecting the next feature to split (instead of using information gain to split the feature). It calculates out of bag error 'a' and then shuffles the values of a particular feature. After shuffling, it again calculates the out of bag error 'b'; higher the difference absolute(a-b), more important the feature is.

Using this, feature importance are calculated and are sorted in descending order of importance. Top 20 features in descending order of feature importance can be seen in figure 9.

4.6 Recursive Feature Elimination [9]

Why do we need it? It makes use of those classifiers that calculate the importance of each feature, while classifying itself; example: logistic regression calculates coefficient for each feature to fit a line or curve, these coefficients are nothing but the

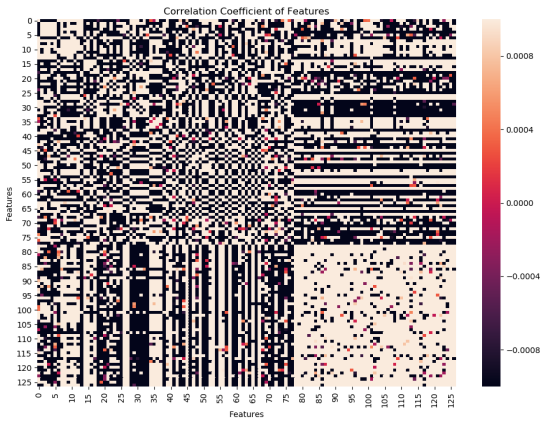

Figure 8: Almost uncorrelated feature pairs (correlation coefficient between -0.001 and 0.001) can be seen as orange-reddish dots

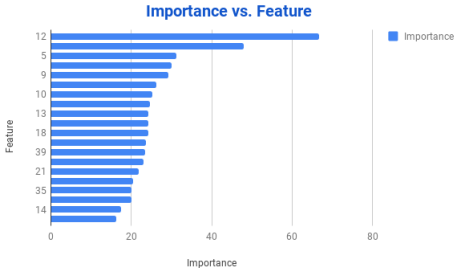

Figure 9: Top 20 features based on the importance determined by Extra Trees

importance of the respective feature. In each iteration, it eliminates the least important feature i.e the feature with the minimum coefficient (in logistic regression), and calculates the coefficients again on the remaining features. Thus, it helps in extracting top *k* features, by eliminating least important feature in each iteration.

Using logistic regression with recursive feature elimination, the rank of each feature is calculated based on its importance (coefficient)

4.7 PCA to get top k components [10]

Why use PCA? In our dataset, we have 127 features i.e 127 dimensions or principal components; some of the dimensions have very less data along them, which increase the complexity of the model by increasing the dimensions of a sample. If we could remove those dimensions that have very less data along them, then our models would be more effective in classifying. PCA helps in identifying the dimensions or principal components, with less information along them.

Using PCA, top k (k ranges from 1 to 126) dimensions or principal components are extracted. This data is then fed to a classifier, which classifies the data by just using top k principal components.

5 Results

5.1 Naive Bayes

Why? Naive Bayes, being a simple model to interpret, gives surprisingly good results *sometimes*, and is extremely fast.

As shown in figure 10, we have used top k (ranging between 1 and 126) components generated by PCA, and fed it to Naive Bayes Classifier i.e every time using top k principal components instead of using all components, which, contain redundant dimensions.

The best quadratic weighted kappa with Naive Bayes is 0.3149 with the data that has top 7 principal components extracted from PCA.

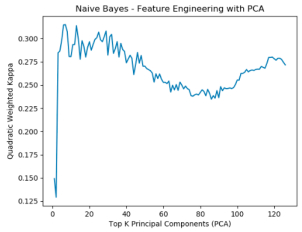

Figure 10: Top K (1 to 126) Principal Components are extracted using PCA and are fed to NB model

5.2 Extreme Learning Machine (ELM)

Why ELM? It is often said that, most of the time, ELM outperforms SVM, by taking just a fraction of the time, SVM takes. ELMs are similar to traditional artificial neural networks, but, they don't tune the weights of all the hidden layers except the weights connecting the last hidden layer and the output layer, thus, ELMs don't need back propagation to update the weights of intermediate hidden layers (all except the last hidden layer), which, makes them too fast (100s of times faster than Artificial Neural Networks).

- As shown in figure 11, Using PCA, top k (1 to 126) principal components are generated, each time these k principal components are fed to ELM.
- We have tuned the architecture of ELM by changing no. of hidden layers, no. of neu-

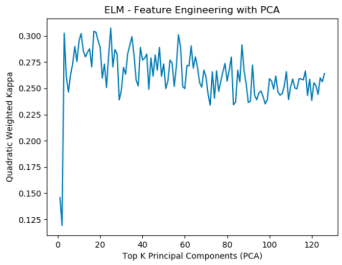

Figure 11: Top K (1 to 126) Principal Components are extracted using PCA and are fed to ELM model

rons in each hidden layer, activation function; increasing no. of hidden layers and neurons, increases non-linearity, thus increasing the variance, which, increases the chance of over-fitting. Linear activation function doesn't give good results as the data is not linearly separable [11].

- The best quadratic weighted kappa with ELM is 0.5047 for the parameters - 2 hidden layers, 1700 neurons each, sigmoid activation.

5.3 Artificial Neural Networks (ANN)

Why ANN? Neural networks have the capability of classifying any meaningful data (with good features); you just need to find the best parameters that give the decision boundary you want.

In addition to the second bullet of ELM, the other parameters like learning rate, no. of epochs are tuned. Small learning rate would slow the process of convergence. High no. of epochs would lead to over-fitting[12][13].

The best quadratic weighted kappa with ANN is 0.5718 for the parameters - 2 hidden layers, 190 neurons each, relu activation function, 7 epochs, 0.02 learning rate.

References

[1] Problem overview, kaggle

[2] Dataset, kaggle

[3] PCA, wikipedia

[4] T-SNE, wikipedia

[5] Correlation Analysis, Google Doc

[6] Power transform, wikipedia

[7] Box-Cox transform in Python, scipy

[8] Extra Trees for Feature Engineering, Google Doc

[9] Recursive Feature Elimination, Google Doc

[10] PCA to get top K Components, Google Sheet

[11] ELM Parameter Tuning, Google Sheet

[12] Neural Network Parameter Tuning - 1, Google Sheet

[13] Neural Network Parameter Tuning - 2, Google Sheet