1. Risk Assessment

The next part of the paper about the risk assessment is organized as follows: First, scope of this work is presented to provide the reader some context about the problem in hand. Next, dataset and the preprocessing processes are explained. Then, the model selection process and the model itself is shown in detail. Finally, results and their interpretations are represented.

a. Scope

From the start, the main objective of this study was to provide some valuable insights to the insurance companies from their data with deep learning methods so that they can increase their profits. In order to achieve this, we did careful research about the insurance sector and saw that a successful risk assessment is one of the most essential, if not the most essential need for making correct decisions while preparing an insurance policy.

b. Dataset

In order to create a data-driven risk assessment model we analyzed our dataset. Dataset was an open-source dataset taken from Kaggle representing what we understood to be a vehicle insurance data according to the provided columns yet it wasn't stated explicitly. Dataset included 53503 data points each representing a customer's data. Data had 30 features and there were no missing values. Accordingly, we had a 53503x30 matrix as our data with some of the features being numeric and some being categorical. The large number of data points and features was promising as large and detailed datasets are usually desired in these types of problems.

Just like many other datasets, we observed some problems about the dataset. Firstly, the features didn't have any explanation. Accordingly, what some features represented were vague. Next, some features wasn't considered valuable for training (for example "Customer ID"). Most importantly, the feature "Risk Profile" was both vague (it had values 0-1-2-3 and what they represented wasn't explained) and undesired as we desired to create a new unsupervised risk assessment model and didn't want labeled data. Therefore, we deleted the aforementioned features.

Next, we had to encode categorical data. Two approaches were used during this process. First, if their arrangement or rank weren't important, we used "One Hot Encoding" where each value for the feature is represented in a distinct column and the one that it satisfies is marked 1 and rest is marked with 0. Next, if their ranks were important, we encoded them according to their following ranks. For example, for driving record, 0 meant "clean" increasing to 4 which meant "DUI (driving under influence). Additionally, some of the features were encoded according to prior knowledge in hand. From our research, we saw that men were slightly riskier for the insurance company. For instance, we encoded gender with 0.45 representing female and 0.55 representing male. Finally, from "Policy Start Date" and "Policy Renewal Date" features which were represented in dates which is non-numeric, a new numeric feature named "Days With Insurance" was created. After these processes, all of our data were numeric.

Finally, in order to avoid some features having higher importance due to their input values, all features are normalized to the range [0,1]. Before this process some features had very high values like such values in a range of [20 001, 149 999] for income levels, where as some features like claim history were in a range of [0,5]. After this normalization process and all the previously mentioned processes, our dataset was a 53503x95 matrix with all of its elements with values in the range [0,1] which was suitable to be fed to a neural network.

c. Model

The choice of unsupervised learning for our model had arisen from the fact that while companies are determining risk profiles, they don't usually have a priori knowledge about risk groups. They create these risk groups according to the a priori knowledge in hand. Considering this, we decided to create a model that will be able to create risk profiles from the insurance data in hand without any other prior risk assessment, in order to align with realistic work scenarios rather than ideal and non-existent situations. With some research about the insurance sector, we decided to use 5 risk groups in this study which are "Very Low Risk", "Low Risk", "Average Risk", "High Risk" and "Very High Risk".

In order to separate our data into risk profiles, we needed a clustering model due to our choice of unsupervised learning approach. From the output of the preprocessing part, we had a large dataset and we needed a model that can handle this large dataset. After some research considering the size of our dataset and the high performances that deep learning methods reach in large datasets, we made an initial implementation of "Unsupervised Deep Embedding for Clustering Analysis" proposed by Xie et al., and saw some promising results. One main idea behind this choice that this model was tested image datasets in the article and image data are usually of significantly larger sizes when compared with regular data, so we believed that this model wouldn't have any problem caused by the size of our dataset.

Deep Embedding for Clustering (DEC) model works as follows: First, a stacked autoencoder with each layer being a denoising autoencoder is trained. Then, the encoded information is fed into standard k-means algorithm to generate the initial clusters. Finally, Kullback-Leibler (KL) Divergence is minimized according to the Soft Assignment clustering (assigning sample to multiple clusters with according probabilities) using Student's t-distribution. In this approach, how well the autoencoder works becomes highly crucial as we are clustering according to the encoded information. For example, clustering may work very well for the encoded information, however if the autoencoder is not performing well and reconstructed information is far off from the original data, we can't rely on the clustering as it depends on a new encoded data which doesn't either represent the original data well or can't be used decoded to reconstruct the original data close enough. Also, the clustering performance of the rest of the algorithm is highly important as it produces the actual clusters. In order to measure the performance of the algorithm, we used "Average Reconstruction Error on Training Data" and "Average Mean Squared Error (MSE)" for the autoencoder to see how well the autoencoder reproduces the encoded data and we used "Silhouette Score" showing how well each point in a cluster is matched to its own cluster, "Davies-Bouldin Index" showing the average similarity between different clusters and "Calinski-Harabasz Index" showing how well the clusters are separated and how dense the clusters are, for the clustering part to see how well the clusters are created. Also, t-SNE was used to visualize the clusters so that we could see how well the clusters were separated and how dense they were. PCA could have also been used for the visualization but t-SNE was preferred as it is more stable giving quite similar results in each run whereas PCA gives very different results in each try. In our tries with the DEC, we saw that clustering performance was well, yet the autoencoder performance had some room for improvement even. As the stacked denoising autoencoder model was a complex model designed for a more complex problem (image clustering problem), we thought that model might had some overfitting issues and therefore we decided to create a custom DEC model for the problem in our hand using a custom autoencoder design combined with the proposed clustering method in the article which uses the encoded data.

For the custom autoencoder, a standard autoencoder is implemented considering the problem's complexity level. To test the model to decide how deep the autoencoder should be, the model is initialized with all the activation functions in encoder part as ReLU (Rectified Linear Unit), and all activation functions in decoder except the last layer as ReLU. The activation function for final layer of the decoder is chosen as Sigmoid function to ensure that the output is in the range [0,1] just like the input. Then, the model is tested with layers of sizes [95,50,25,10] in encoder and accordingly [10,25,50,95] in decoder. The initial criterion was MSE loss and optimizer was ADAM (Adaptive Moment Estimation) with learning rate 0.001. With this setup we saw that average reconstruction error and average MSE for autoencoder were around 0.06 but clustering results were not satisfactory. Silhouette score is in the range [-1,1] where 1 represents good cluster assignments, 0 means cluster assignments are in border, and -1 means bad cluster assignments, Davies-Bouldin index is in range [0,∞] where results near 0 are considered good as it means similarity between clusters are very small, and Calinski-Harabasz index doesn't have a range but higher values mean a great separation and high density of clusters so higher values are desired. In an example for this setup, Silhouette Score was 0.299, Davis-Bouldin Index was 1.048 and Calinski-Harabasz Index was 29781.81 which is not a satisfactory clustering.

Theoretically, encoding to smaller feature sizes should increase the clustering performance as it is easier to separate the data with small feature dimensions compared with the data that has high feature dimensions. However, when we encode the data to smaller dimensions more information is lost in the autoencoder. Therefore, there is a trade-off between the clustering performance and autoencoder performance. At this study, we are desiring to find an optimal point where both autoencoder performance and clustering performance

With these ideas and similar tries, it was observed that optimal model included an autoencoder with sizes [95,50,25,10,5,2] in encoder and [2,5,10,25,50,95] in the decoder, creating an autoencoder that is able to reduce the 95 features into 2 features, and then rebuild it to 95 features from these 2 encoded features. At this setup, autoencoder performance dropped to around 0.081 but clustering performances increased significantly. The drop of autoencoder performance around 0.02 may seem very low but considering that the input and output is scaled in the range [0,1] this is an important drop and the performance of the autoencoder should be increased. In order to do it, first we tried SGD (Stochastic Gradient Descent) with different learning rates and also tried ADAM with different learning rates and saw that ADAM with learning rate equals to 0.003 gives the best performance for our problem. Also, we tried different loss functions like Cross-Entropy Loss and Binary Cross-Entropy Loss alongside MSE Loss, and in terms of clustering performance we see at the end of the whole model, all gives very close results with Binary Cross-Entropy giving the best results. However, Cross-Entropy and Binary Cross-Entropy didn't use the same unit as input and output so we weren't able to observe how much the reconstruction loss was on these functions. We could just observe the convergence graph. Therefore, as the results are very close in terms of clustering, we decided to continue with MSE Loss for our model since we could easily observe the performance of autoencoder with the "Average Reconstruction Error On Training Data" metric.

During the training of these previous steps, we occasionally observed that improvements stopped during the iterations for the epochs. With some research, we observed that this might be caused by dying ReLU problem which refers to some neurons becoming inactive and only give zero as output regardless the input. In order to solve this and increase the overall performance we replaced ReLU with Leaky ReLU and ELU respectively. We saw that Leaky ReLU gave better results, and ELU gave even better results. As a result, we decided to replace the ReLUs with ELUs. We also tried the model with dropout layers between layers with probabilities p=0.2, p=0.1, and p=0.05 to see if our model is suffering from overfitting and improve the results if it does. However, contrary to our expectations, it worsened the autoencoder performance and increased the average reconstruction error so we decided not to include any dropout layers. With this try of dropout layers, we saw that we were not suffering from overfitting problem. With the goal of improving the autoencoder performance, we also tried putting Batch Normalization between layers as it is a commonly used process. However, the computation time increased by a great amount and yet performance did not increase, even dropped a little bit. Accordingly, we also decided not to use batch normalization just like the dropout layers.

After all these models, our custom DEC model formed as follows: A standard autoencoder with encoder layers of size [95,50,25,10,5,2] and decoder layers of size [2,5,10,25,50,95]. The activation functions used are ELUs instead of the last layer of decoder which is the Sigmoid function to ensure that the output is in the range [0,1]. Loss criterion is the MSE (Mean Squared Error) Loss and the optimizer is ADAM with learning rate equal to 0.003. No batch normalization or dropout layers is used. After the custom autoencoder part, the model continues the same as the DEC model. First, standard K-Means algorithm is used with the encoded information as its input to find the initial clusters. Then, KL Divergence is minimized according to the Soft Assignment clustering using Student's t-distribution to find the optimum clusters. Using this model, the following results represented in Figure 1-4 are obtained.

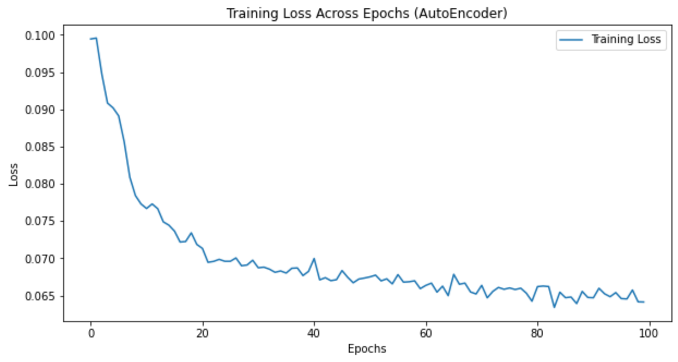
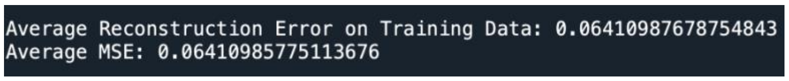
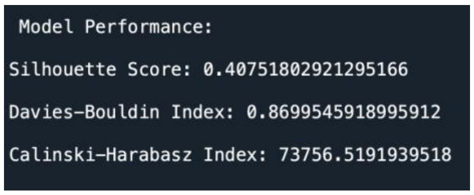


Figure 1: AutoEncoder Convergence


```
Average Reconstruction Error on Training Data: 0.06410987678754843
Average MSE: 0.06410985775113676
```
Figure 2: AutoEncoder Performance


```
 Model Performance:

Silhouette Score: 0.40751802921295166

Davies–Bouldin Index: 0.8699545918995912

Calinski–Harabasz Index: 73756.5191939518
```
Figure 3: Model's Clustering Performance

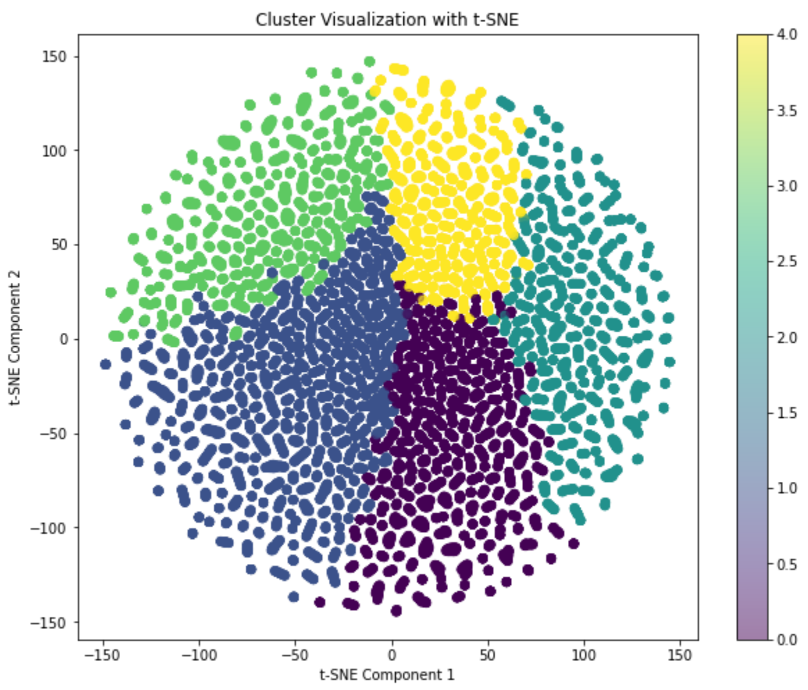


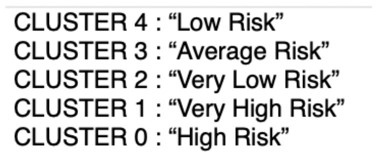
Figure 4: Cluster Visualization with t-SNE

From the t-SNE visualization and model performance metrics we observe that clusters are greatly separated with small similarity between them, each cluster is dense and data points match to its own cluster well. Therefore, we can say that clustering is good. When we look to the Autoencoder performance metrics, we see that with the improvements we made in the custom model we reached to 0.0641 average reconstruction error on training data. This is a very good result for an encoding to 2 features considering that in the initial try with encoding to 10 features the average reconstruction error on training data was around 0.06. Overall, we believe that we found a highly optimal model providing both a good clustering performance and autoencoder performance, when the tradeoff between them is taken into consideration.

d. Interpretation Of The Results

After producing the clusters, we needed to label them as they were unlabeled. This is a great point to say that actually after this point, a person working in the insurance sector will most likely to label these clusters better than us due to their experience, yet in order to provide our study with some final results we also labeled these clusters after some post-processing processes.

At first, we calculated the average of each future distinctly for each future too see whether we can label these clusters according to their averages. However, according to the central limit theorem, distribution of large data converges to normal distribution and we had the same problem. Most of the features' averages were around 0.5. Therefore, to solve this problem, we calculated the quantile range between 0.4 and 0.6 for each feature for each cluster to find the data points that represent their clusters best and obtained the average of each future between these data points. And then we analyzed this post-processed data and decided for cluster labels. Figure 5 below shows the final results.



CLUSTER 4 : "Low Risk"
CLUSTER 3 : "Average Risk"
CLUSTER 2 : "Very Low Risk"
CLUSTER 1 : "Very High Risk"
CLUSTER 0 : "High Risk"

Figure 5: Cluster Labels