Project: Automated risk assessment of PyPi packages using unsupervised machine learning

Date: 26/10/2023

Student Engineer:
Hari Markonda Patnaikuni

Student ID:
n10789511

Supervisor:
Dr. Gowri Sankar Ramachandran

Table of Contents

Acknowledgements

A special thank you to Dr. Simon Denman for permitting me to use his code for the two clustering algorithms from his CAB420 practical solution.

1.1.1 Abstract

This project explores the capabilities and limitations of unsupervised machine learning and how it can be used to automate the risk assessment for open-source packages from the Python Package Index via their metadata in the form of scores out of 1. Six features were considered when calculating the scores of risk and reliability for these packages. As a result, using the scores of these package's metadata the scores of risk and reliability were generated for 8000 packages.

These risk and reliability scores were then compared to a validation dataset and analysed. It was found that the data distribution for the risk and reliability scores found using the K-Means clustering model resulted in trends that were the inverse of the validation risk and reliability scores. To remedy this drastic change in trends observed the risk and reliability scores found using the machine learning approach were adjusted for a certain group of packages via a adjustment factor. This aided in reducing the drastic difference between the scores for the clustering and validation.

1.1.2 Introduction

Due to the current state of Python development and the dependency on open-source packages to implement solutions to our software's requirements, many people tend to overlook the security risks associated with using these packages (Markonda Patnaikuni, 2023). This has become an important area of research to the extent that the US government has signed an executive order on improving the nation's cybersecurity including the enhancement of software supply chain security (Markonda Patnaikuni, 2023) (Executive Order on Improving the Nation's Cybersecurity 2021 s 4).

Many industrial applications use open-source PyPI packages, potentially exposing sensitive user data to malicious attackers (Markonda Patnaikuni, 2023). The general objective of this project is to utilise unsupervised machine learning to automate risk assessment for PyPi packages at a large scale (Markonda Patnaikuni, 2023). The research question for the project is "Can unsupervised machine learning be used to automate risk assessment for PyPi packages at a large scale?" (Markonda Patnaikuni, 2023).

This project aims to use data from PyPi packages to analyse and return an automated risk assessment for the package analysed (Markonda Patnaikuni, 2023). The unlabeled data comes from PyPi warehouse which has package information and Github which has information about contributors and how they are building the package (Markonda Patnaikuni, 2023). The risk assessment consists of a reliability and risk score for each package.

This report primarily focuses on the final results of the project with emphasis on the method followed to pre-process the unlabeled data and start inputting the training and testing data

into two types of clustering algorithms. The first algorithm is K-Means, and the second algorithm is Gaussian Mixture. In addition to this, the K-Means algorithm was chosen to generate the final risk assessments for the packages which consists of a risk and reliability score for each package in the test dataset.

1.1.3 Literature Review

This review of literature focuses on data analysis and machine learning techniques that can be used to identify risks within PyPi packages via their metadata. In addition to this, the features which are considered risks are also focused on. The sources analysed use PyPi and other open-source libraries such as npm to find the risks within the packages(Markonda Patnaikuni, 2023).

The first and possibly the most important key theme found in most of the papers that utilise machine learning to find risks within an open-source library is the type of machine learning used by the researchers. In papers such as “On the Feasibility of Supervised Machine Learning for the Detection of Malicious Software Packages” and “Practical Automated Detection of Malicious npm Packages” supervised machine learning is used to identify malicious software packages (Sejfia & Schafer, 2022, p.1) (Ohm, Boes, Bungartz, & Meier, 2022, p.3) (Markonda Patnaikuni, 2023).

This relates to the research question which focuses on an unsupervised machine learning approach to find the risks within the metadata of a PyPi package. Hence, it can be said that the lack of unsupervised machine learning in both papers makes the proposed research question more important to answer due to the lack of unsupervised machine learning research (Markonda Patnaikuni, 2023).

Another research paper titled “Unsupervised Feature Selection for Outlier Detection on Streaming Data to Enhance Network Security” focuses on using unsupervised machine learning to find outliers in streaming data (Heigl, Weigelt, Fiala, & Schramm,2021). This paper shows that unsupervised machine learning can be used for the feature selection of data based on outlier detection. Hence, this feature selection could then be adapted to this project to find features within PyPi package metadata to find risks within the packages (Markonda Patnaikuni, 2023).

In addition to this, the type of datasets used in these papers includes data that classifies certain packages as malicious already. Hence, the supervised nature of machine learning is used. In this project, the data used is purely metadata about hundreds of thousands of PyPi packages which need to be analysed via an unsupervised machine-learning algorithm to look for risks in

these packages. Thus, the data used in this project is unlabelled and cannot be used with any other form of machine learning other than unsupervised (Markonda Patnaikuni, 2023).

Another key theme found in the papers is the types of features to be extracted from the datasets. The types of features that are extracted from the dataset are important in classifying the risk associated with the PyPi packages (Markonda Patnaikuni, 2023).

For example, in the paper titled “On the Feasibility of Supervised Machine Learning for the Detection of Malicious Software Packages”, the features extracted by the researchers include npm-specific features such as, “whether an installation script opens a network connection”, and if there are suspicious strings in the package such as “/etc/shadow or .bashrc” (Ohm, Boes, Bungartz, & Meier, 2022, p.3) (Markonda Patnaikuni, 2023).

Hence, the types of features extracted by the machine learning model play a critical role in the model being able to automate risk assessment of the PyPi packages. The project may have to extract features from the metadata of a package which may include features such as the date of the last commit, the number of contributors to the package, the number of stars for the project on GitHub, etc. In addition to this, the source file of the package may have to be analysed to look for any comments, functions, variables that have suspicious names, or even combosquatting within the source files of a package (Markonda Patnaikuni, 2023).

A type of classification algorithm that might be used in the project is clustering via K-Means which is an unsupervised machine learning algorithm that can be used to extract suspicious domain names in npm and PyPi packages as discussed in the paper titled “A Survey on Common Threats in npm and PyPi Registries” (Kaplan & Qian, 2021, p.21). This general idea could be adapted to this research project for multiple features of a PyPi package (Markonda Patnaikuni, 2023).

Some other types of unsupervised machine learning algorithms that might be used in the project may be clustering, hierarchical clustering, and Gaussian Mixture Modelling. These three models have been used in the paper titled “Design and Evaluation of Unsupervised Machine Learning Models for Anomaly Detection in Streaming Cybersecurity Logs” for anomaly detection in streaming cybersecurity logs. These applications of unsupervised machine learning are like the project's aim of finding risks in PyPi package metadata (Sanchez-Zas., Larriva-Novo, Villagra, Rodrigo, & Moreno, 2022). Hence, these algorithms may also be able to be adapted to this proposed project (Markonda Patnaikuni, 2023).

Another key theme found during research is the process of empirically analysing the data of PyPi packages and illustrating the data in an easy-to-understand way. This can be done by grouping the packages based on any parameter. This analysis was shown throughout the paper titled “Empirical Analysis of Security Vulnerabilities in Python Packages” (Alfadel, Costa, & Shihab, 2023). For example, the packages can be grouped by the most recent commit on GitHub. In general, it can be suggested that packages that have a relatively recent commit are more likely to be safe (Markonda Patnaikuni, 2023).

Whereas packages that have not had any recent commits after their initial commit may be more likely to be either a legacy package or a malicious one. Hence, this analysis technique could be used as a data grouping technique for a large amount of PyPi packages. The packages could be grouped based on a particular parameter and then these groups of packages can be evaluated in batches in the machine learning algorithm (Markonda Patnaikuni, 2023).

The general methodology for the analysis of PyPi packages as evident in the research papers titled “Design and Evaluation of Unsupervised Machine Learning Models for Anomaly Detection in Streaming Cybersecurity Logs” is to first get data to analyse (Sanchez-Zas., Larriva-Novo, Villagra, Rodrigo, & Moreno, 2022). This data is usually found in open-source APIs focused on the PyPi package index. For this project, the data is from both the PyPi warehouse and GitHub (Markonda Patnaikuni, 2023).

After this step, the data is pre-processed to allow for the correct scaling of the data and better performance of the yet-to-be-developed machine learning algorithms. The next step in the analysis is to create an algorithm to analyse the data. After this, the results of the analysis are presented in graphs and tables to evaluate key trends found in the results of the analysis. As a result, this general method can be used for this project (Markonda Patnaikuni, 2023).

Another key theme that was evident in the review of the literature was that the machine learning algorithms used in papers are usually supervised. This is a gap in the current state of research as there has not been much research regarding the PyPi package's risks and how to find these risks using unsupervised machine learning. As this research project intends to use unsupervised machine learning the scope for filling a gap in the current research field does increase. This may introduce a new methodology that can be used as a starting ground for future projects with different open-source libraries such as npm (Markonda Patnaikuni, 2023).

The research gaps addressed by this project include the use of unsupervised machine learning for open-source PyPi package security and the generation of an automated risk assessment based on an unsupervised machine learning model. This project uses a dataset of unlabelled packages to group these packages into clusters. Based on the package’s distance to its cluster a score of reliability and risk is generated for all packages.

1.1.4 The Data

The dataset used for this project is a CSV file containing PyPi package metadata. This file contains the metadata for over 20000 packages with 6 features of metadata. All features within the dataset are in the scale of a score out of 1. The process to calculate the score is an in-house method that will become open source in the near future.

The first feature in the dataset is the software maturity score which is calculated based on the age of the package. The higher this score is the more mature the package is. Hence, the maturity score increases as the age of the package increases.

The second feature in the dataset is the freshness score which is based on the most recent update. Like the software maturity score this score is also based on age, however, it is different as it looks to see when the package had its last GitHub update. This score also has a proportional relationship to the age of the last update hence, the more recent the update is the higher the score.

Another feature found in the dataset is the accessibility score which is based on the availability of documentation websites related to the package. This score also increases as more documentation and websites are available referencing the package and its use.

The community engagement score is another feature found in the dataset which is based on the number of contributors a package has on its GitHub page. The more contributors a package has the higher its community engagement score.

The dataset also contains a feature called the vulnerability score which is based on the author's email status. An inactive email status increases the vulnerability score. This feature is an important feature due to the potential for hacking if an email is inactive. If the email status is inactive, then a potential hacker can buy the domain name and access the email of the author and hence the GitHub account of the author. Then the hacker may push malicious code changes to the package and cause several cybersecurity incidents to occur.

Finally, the popularity score is based on the number of GitHub stars a package has on its GitHub page. This score increases as the number of GitHub starts increases.

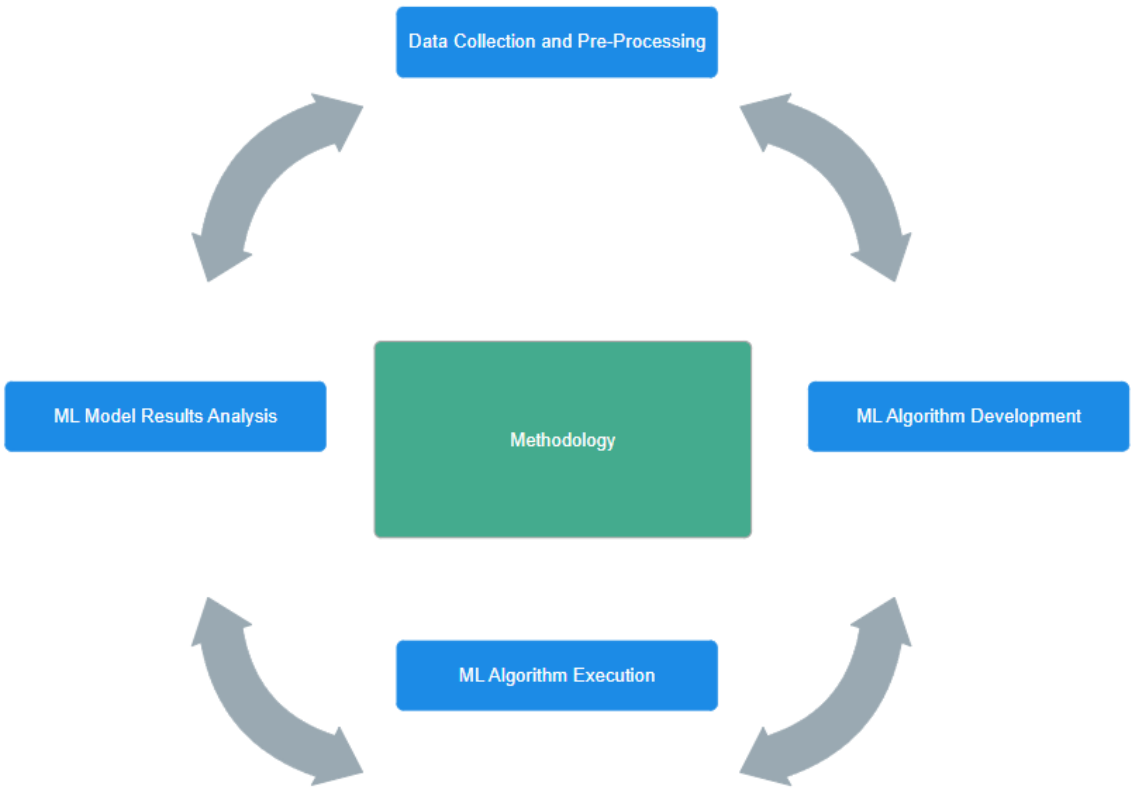
As a result, due to the importance of the above features and how they can predict if a package is risky or not it is safe to say that they should be used when developing a score of reliability for a PyPi package.

1.1.5 Methodology

The methodology used consisted of first pre-processing the dataset. Then the dataset was split into training and testing data. After this, the datasets were input into an unsupervised machine learning algorithm. The output of this model includes a clustering graph and another histogram that shows information about each cluster. In addition to this, the model also prints the overall risk and reliability score for each package based on its distance from its cluster and its cluster’s characteristics.

This methodology is illustrated in the below figure. It can be said that this general methodology is a standard methodology in the field of machine learning and hence was used due to the main task in this project being a machine learning problem in nature.

Figure 1: Methodology Flow Diagram

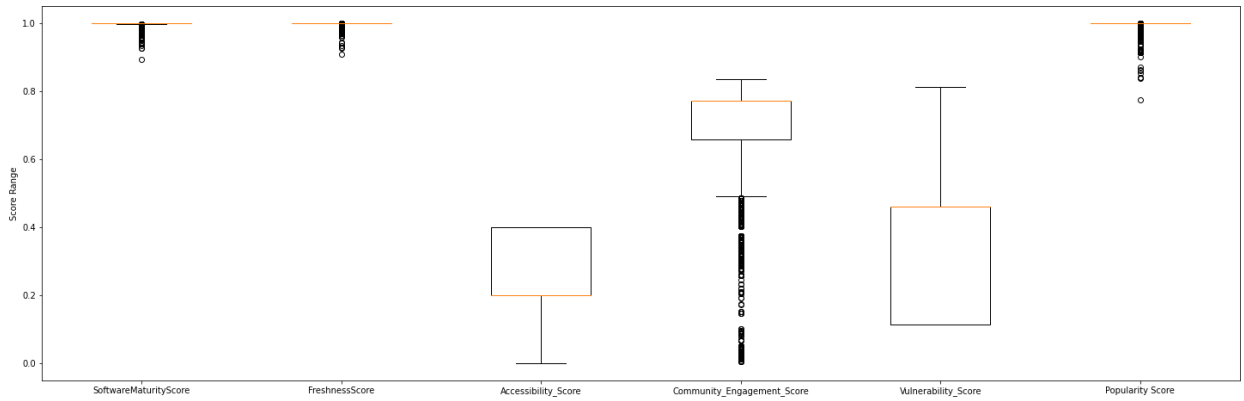


1.1.6 Pre-processing of data

The data of the PyPi packages are located within a CSV file. This file is loaded into the 'ipynb' file via the 'pandas.read_csv function which is a function imported from the pandas library. After this, the data is transformed so that the testing set of the data contains information from rows 0 to 8000 with only information from columns 1, 2, 3, 4, 5, and 6.

A similar approach is followed for the training dataset except only the rows from 8000 to 16000 are included. Neither the training nor testing dataset were standardised due to the features used being all on the same scale of a score out of 1.

Figure 2: Boxplot of the test dataset



1.1.7 Clustering models used

The clustering models used in this phase of the project are from the Keras library which includes models that can be called for K-Means clustering and Gaussian Mixture clustering. The training and testing datasets are input into the main clustering analysis models. Then the clusters are plotted on a graph and the characteristics of these clusters are illustrated via histograms.

1.1.8 Assumptions, Errors, Ethical Considerations, and Scope for Improvement

An assumption that was made in this project is that all the features within the dataset are represented numerically. This ensures that the scale for these scores of each feature is the same and that the clustering models work correctly. Another assumption that was made is to not standardise the training and testing datasets. This decision was made due to all the features within the dataset being in the same scale hence, not needing standardisation.

One of the most important assumptions made in this project is that it is assumed that the centres of all clusters found by the clustering models are used to judge the reliability of a package based on its distance to its cluster. Hence, the reliability is inversely proportional to the package's distance to its cluster.

A major error that was found in this project is the variability between the validation and K-Means risk and reliability scores. This error is most likely occurring due to the way these scores are calculated.

Due to the open-source origin of the dataset, the ethical considerations were not of much importance for this project. However, one feature from the dataset that may have some ethical issues is the vulnerability score. This is because this score is based on the email status of the author of the package. Due to this information being a prime commodity for hackers to exploit there is an ethical consideration that must be made when handling

information of this nature. However, for this project, the email status is only represented as a number out of 1 in the dataset and the risk assessment does not explicitly say which emails are inactive.

An area for improvement would be to use more packages when running the clustering models. This would result in a more accurate score for the reliability and risk scores. In addition to this, this would most likely reduce the discrepancy between the validation data's scores and the clustering approach.

1.1.9 Results and Analysis

The dataset used for all the tests include 6 features and 16000 packages. The below clustering figures show the data distribution for three clusters. The features used include the software maturity, freshness, accessibility, community engagement, vulnerability, and popularity score of each package. As mentioned previously, all these features are a score out of 1 and are not standardised as they are of the same scale.

The histogram shows an average score for each feature within its cluster. It can be seen in the histograms for each cluster that the defining features that influence the cluster's separability are the vulnerability score and the community engagement score to a lesser extent. In particular, the vulnerability score which is a score based on whether the author of the package has a valid email address is a key feature that seems to contribute to the relative risk of open source packages.

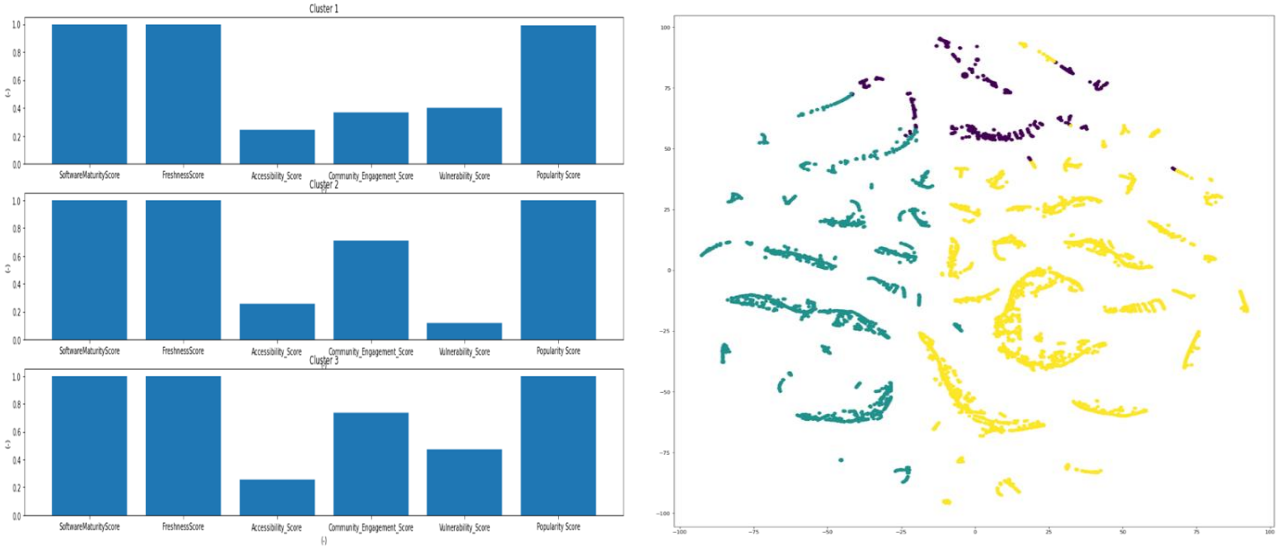
In a real-world scenario, this makes sense as well as the email address of an author may be inactive or expired. This may result in a scenario where a hacker buys the domain to this email address and gains access to the GitHub account of the original creator of the package. Following this the hacker may push changes to the package that contains source code changes that can compromise the security of a user's PC.

The community engagement score also influences the cluster a package belongs to. This score, which is based on the number of contributors to the package on GitHub can also be considered as an important feature within the dataset. In the case where the community engagement score is around 0.7 for clusters 2 and 3, this can be considered as decreasing the risk of the package as more community members are contributing to the package's source code.

Whereas, for cluster 1 the community engagement score is around half at 0.35. This could mean that the risk of the package is increased due to the lack of community members contributing to the package on GitHub. Another reason as to why the community engagement score is low may be due to the package not being well known and the author not allowing for open source contribution by the community.

The graph of the three clusters shows that they are mostly separated with the yellow cluster having the most packages. The yellow cluster belongs to cluster 1, the turquoise cluster belongs to cluster 2, and the purple cluster belongs to cluster 3.

Figure 3: K-Means 6 Features and 3 Clusters



The below figures show the same 6 features used for a K-Means clustering analysis but with 2 clusters instead of 3. In this case, the clusters are mainly separated based on the average vulnerability score of each package. Cluster 1 has an average vulnerability score of around 0.5 whereas cluster 2 has an average vulnerability score of around 0.1. This is translated to the clustering graph with cluster 1 belonging to the purple cluster and cluster 2 belonging to the yellow cluster.

When compared with Figure 3, figure 4 has around the same community engagement scores for clusters 1 and 2 whereas, for Figure 3 there was variation in the scores. This means when the number of clusters is 2 for the K-Means model the vulnerability score seems to be the most important and varying feature.

Figure 4: K-Means 6 Features and 2 Clusters

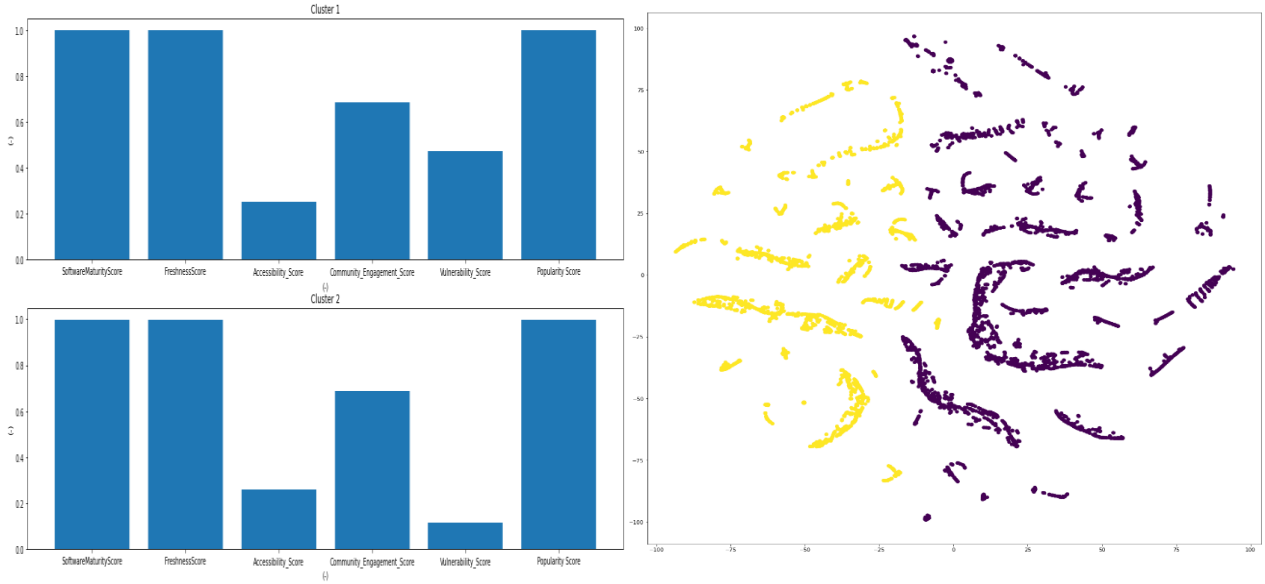
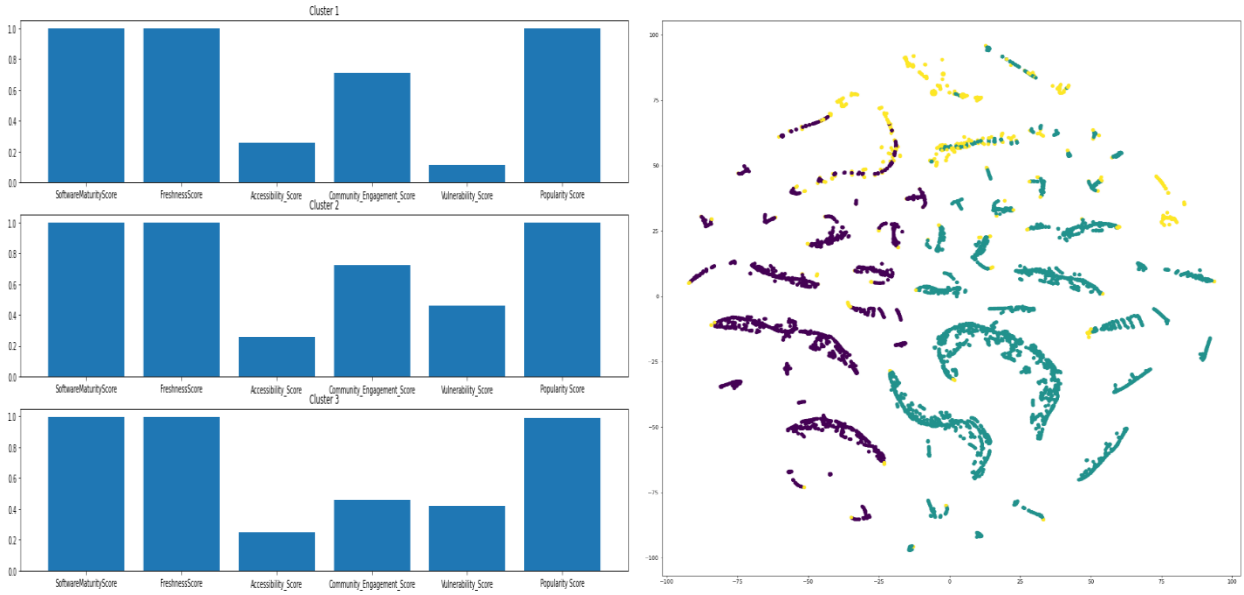


Figure 5 shows that the GMM clustering model results in more overlap between the three clusters when compared to the K-Means model. Like the K-Means model the main feature that contributes to the separation between the clusters is the vulnerability score hence, again reaffirming its key role in separating each cluster from another cluster. Due to the GMM model having more overlap in its clusters the decision was made to not generate reliability and risk scores for the packages using the GMM model.

Figure 5: GMM 6 Features and 3 Clusters



The below figure shows the data distribution for the risk scores generated by the K-Means clustering algorithm for 8000 packages. These risk scores were found by finding each package's distance to its assigned cluster. For this case, the total number of clusters used is 2. The general trend observed in the below histogram is that most of the packages have a risk score between 0.05 to 0.186 which in a percentage representation is 5% to 18.6%. In addition to this, there are around 5 packages with a risk score of more than 60%. As a result, it can be said that most packages have a low risk score when using each package's distance to its cluster as the main measurement of risk.

Figure 6: Histogram of Risk Scores for 8000 packages

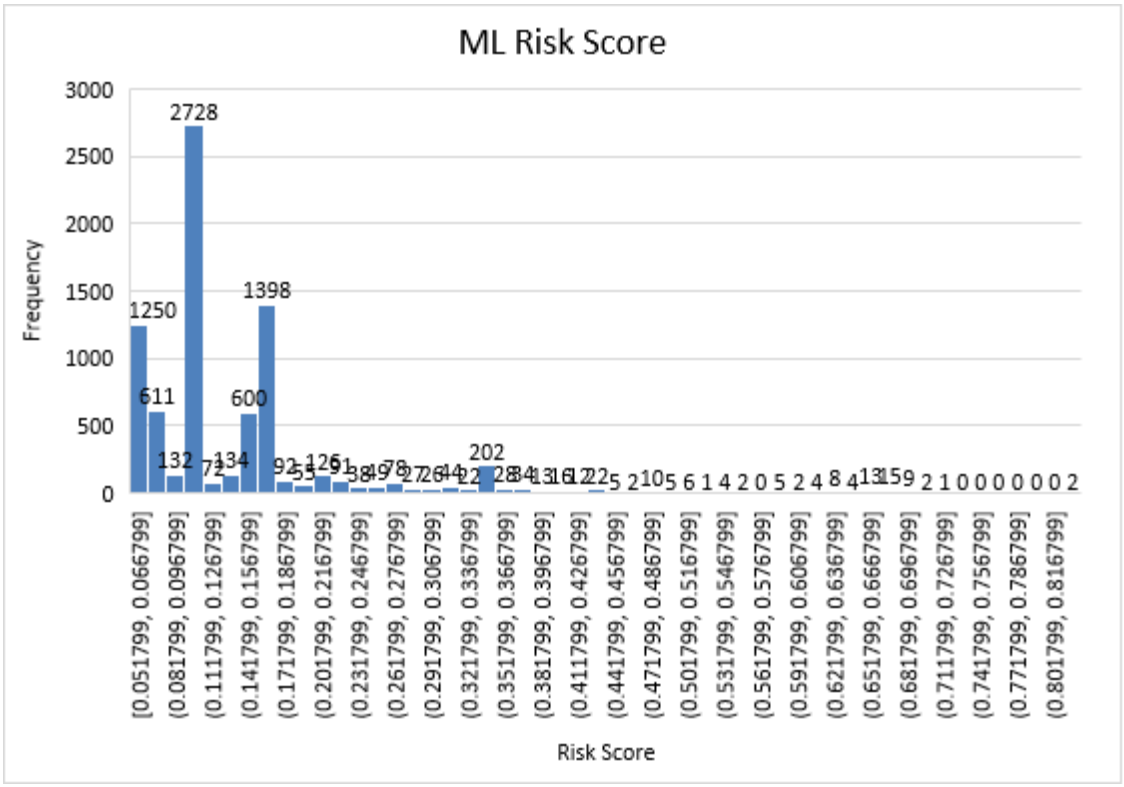
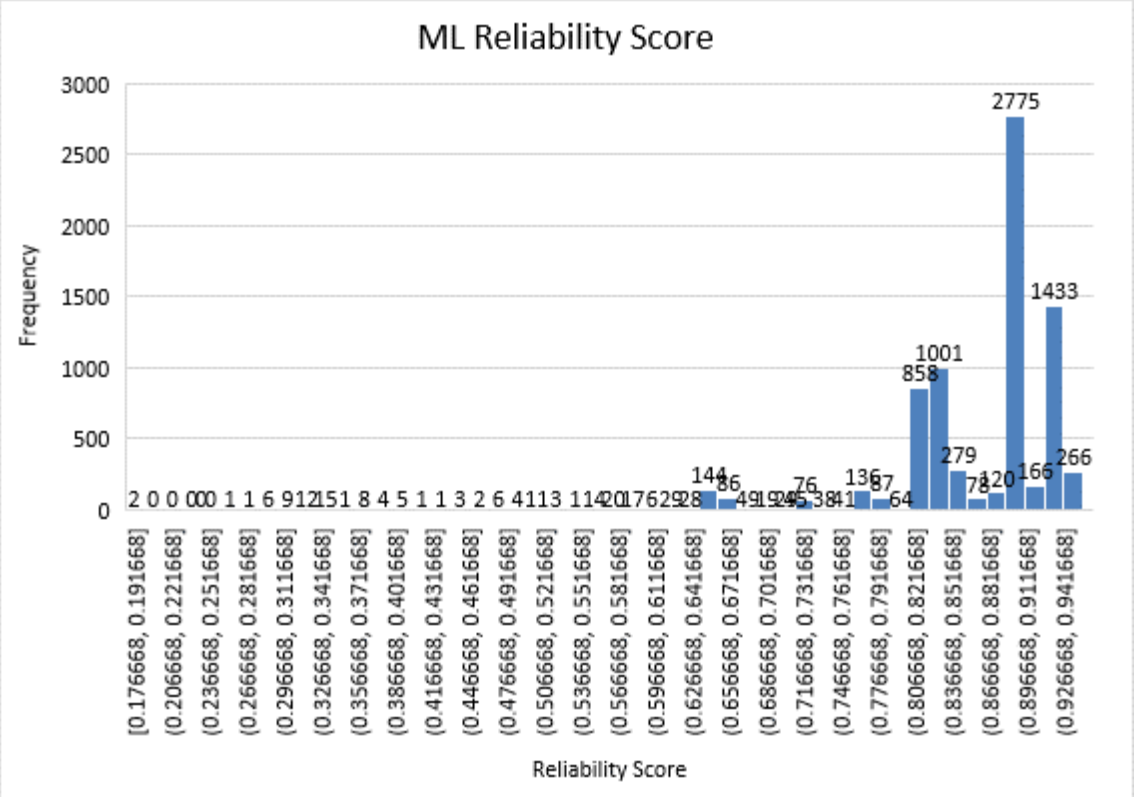


Figure 7 shows the data distribution for the reliability scores of 8000 packages. The reliability scores were calculated by subtracting the risk score from 1 for each package. The general data distribution illustrated in the below histogram shows that most packages have a reliability score between 80% to 94%. Around 4 packages have a reliability score that is below 30%. These results indicate that most packages have a relatively high reliability score when using the package's distance to its cluster as the main measure of reliability.

Figure 7: Histogram of ML Reliability Scores for 8000 packages



The below histogram shows the data distribution for the number of clusters assigned to cluster 1 and 2 using the K-Means clustering model with 2 clusters and a random state of 59. It can be said that cluster 1 has a larger number of packages assigned to its cluster when compared with cluster 2. This is what the frequency axis is referring to hence, cluster 1 has 4986 packages assigned to it and cluster 2 has 3014 packages assigned to it. As discussed previously the main reason for this data distribution is due to the average vulnerability score for each cluster.

Figure 8: Histogram of ML Packages Cluster Distribution

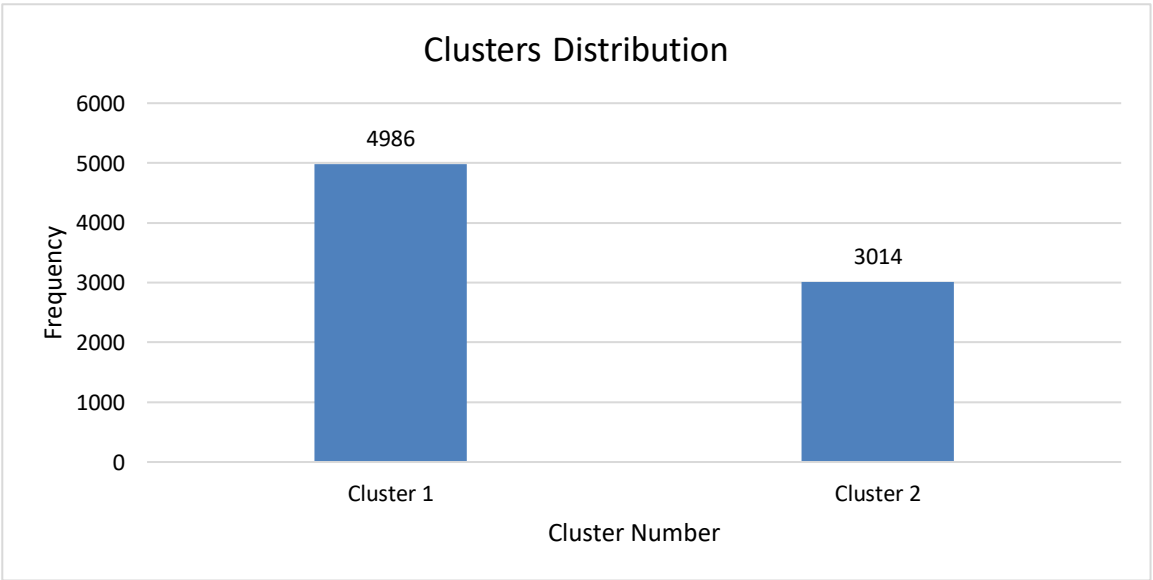
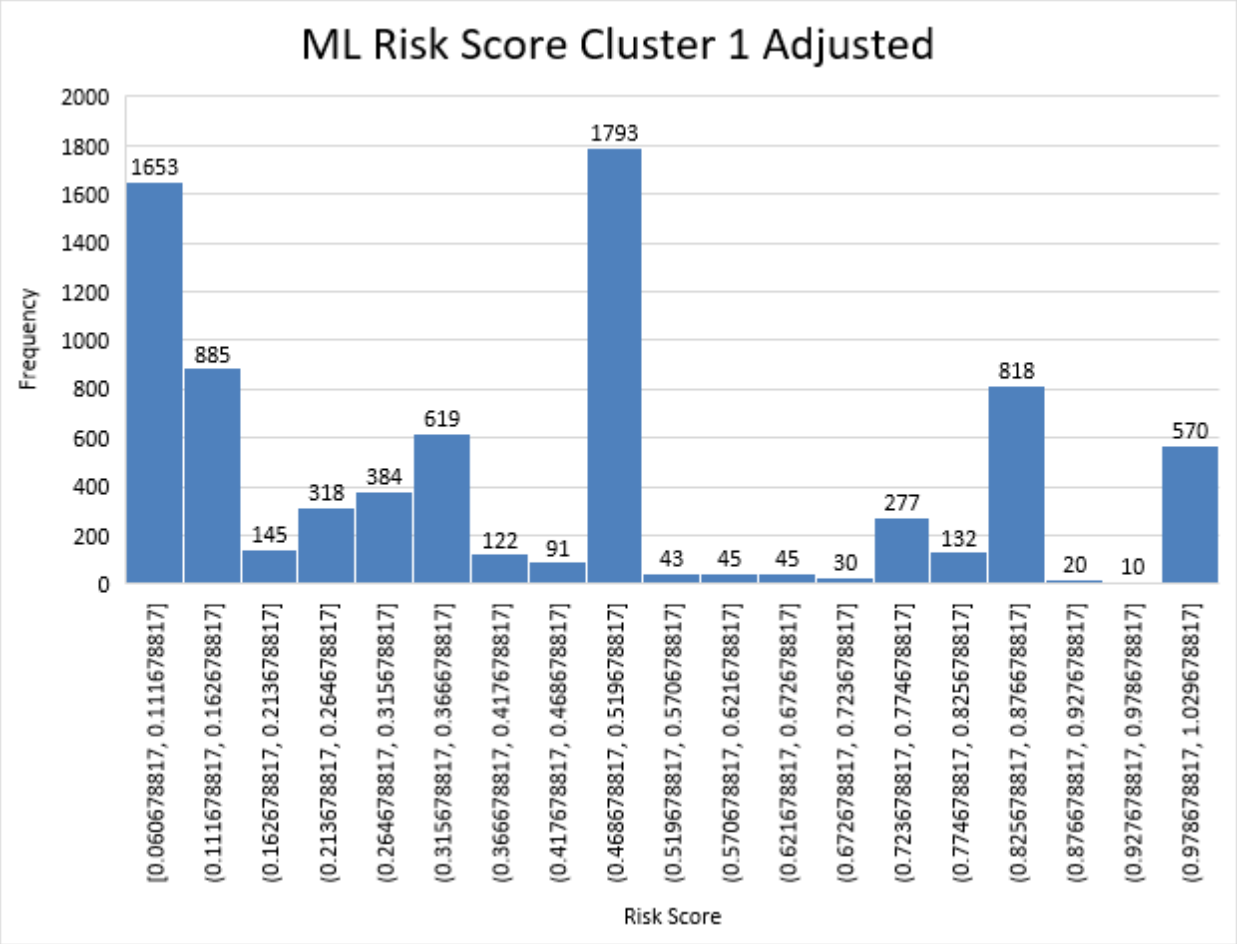


Figure 9 shows the data distribution for clusters 1 and 2 after packages in cluster 1 risk scores were adjusted due to the vulnerability score being higher for cluster 1. This resulted in a major change in the data distribution observed when compared to the data distribution observed in Figure 6. It can be said that more packages have a higher risk score when compared to Figure 6's data distribution. This adjustment to the risk scores for cluster 1's packages consisted of multiplying all risk scores by 5 as the vulnerability score on average 5 times higher than cluster 1.

Figure 9: Histogram of ML Risk Scores with Cluster 1 Adjusted



Similarly, for Figure 10 the reliability scores data distribution has changed drastically when compared to Figure 7. This is due to the adjustment made to the risk scores which then also adjusts the reliability scores for packages in cluster 1. It can be said that the more packages have a lower reliability score whereas in Figure 7 most packages had a relatively high reliability score.

Figure 10: Histogram of ML Reliability Scores with Cluster 1 Adjusted

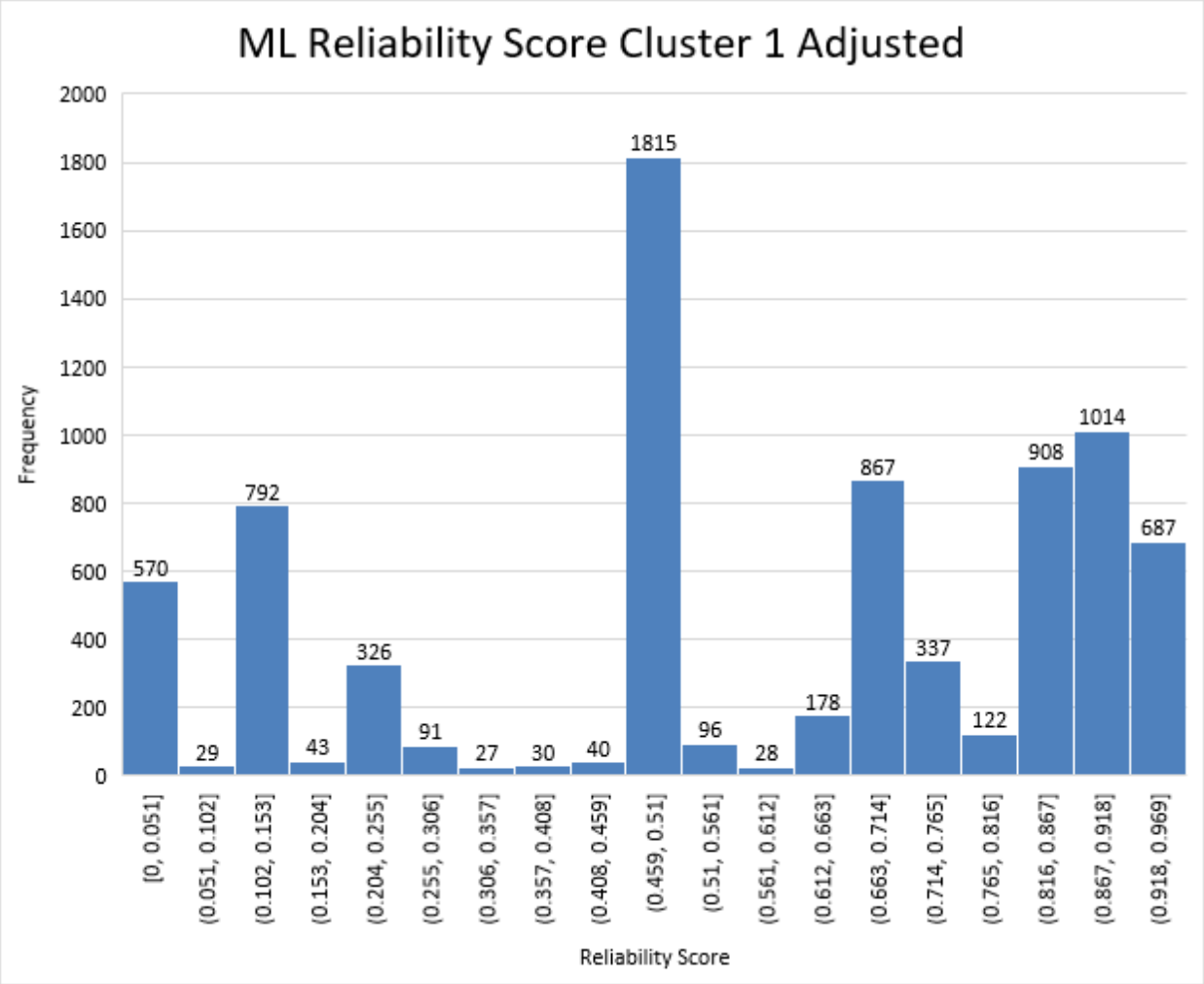
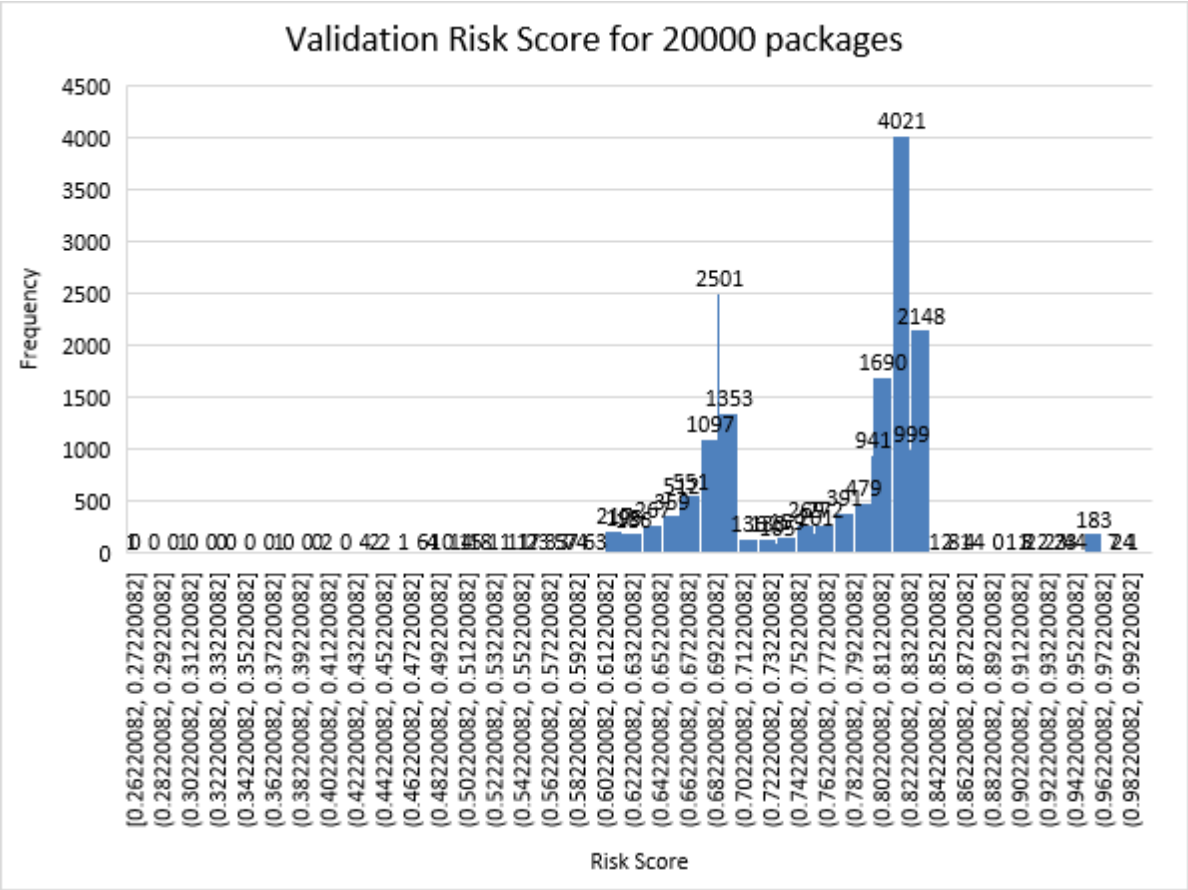


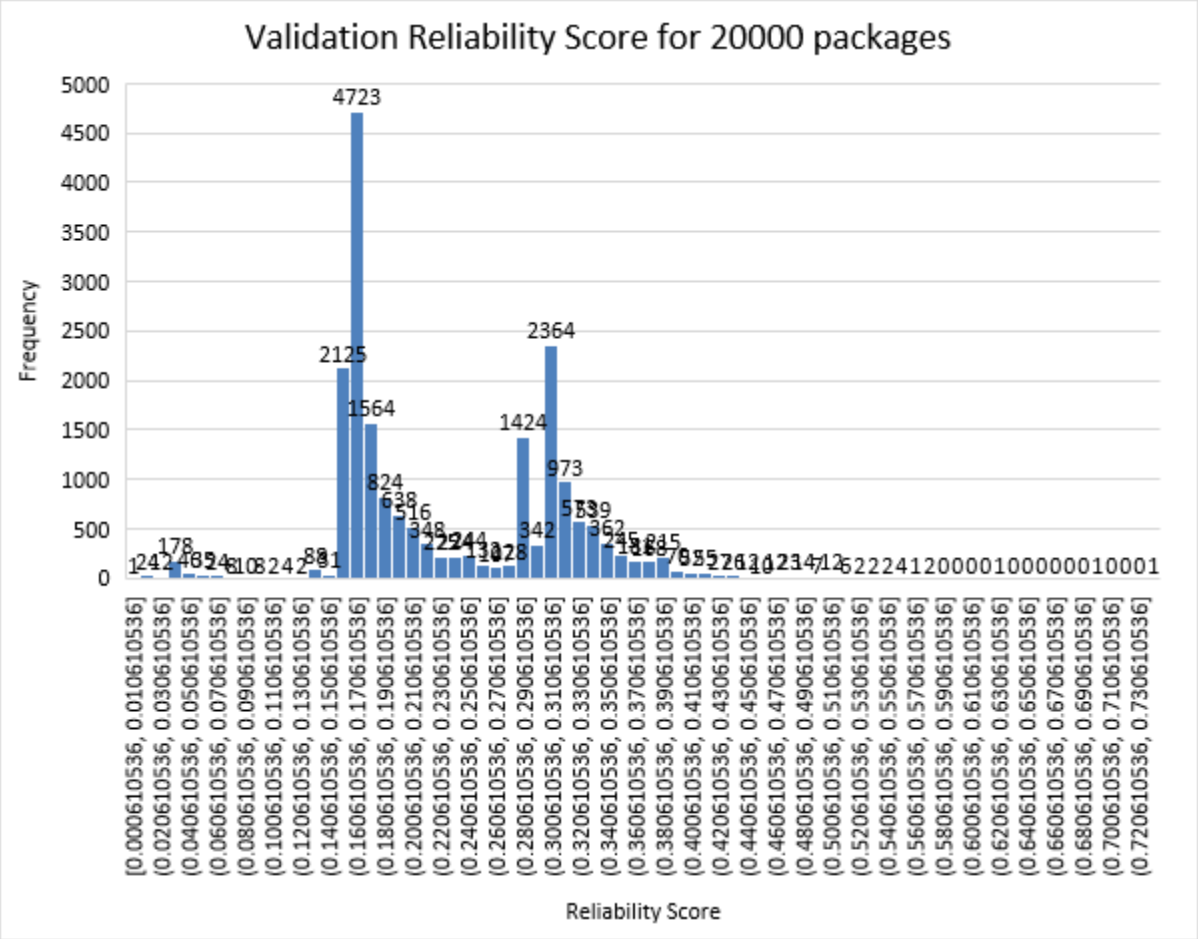
Figure 11 shows the data distribution for the validation risk scores for 20000 packages. The general trend observed shows that most risk scores are in the range between 60% to 85%. It can be said that this is a major difference in risk scores when compared to the risk scores found using the K-Means model. To some extent this is expected due to the way the risk scores are calculated for both methods. However, the stark difference in the trends observed will need to be discussed further.

Figure 11: Validation Risk Score for 20000 packages



Finally, Figure 12 shows the distribution of data for the validation reliability scores. The trend of most packages having a reliability score between 14% to 35% is observed. This score of reliability was calculated by subtracting the risk score from 1. It can be said that these results show that the reliability score distribution between the K-Means and validation methods are vastly different in terms of data distribution. This difference in data distribution will be discussed extensively in the discussion.

Figure 12: Validation Reliability Score for 20000 packages



One of the key findings of the above results is that both clustering models can form distinct clusters of sizes 2 and 3. In addition to this, the models can find distinct characteristics of these clusters and plot them on a histogram for each cluster. These distinct characteristics relate to the most important features that determine whether a package is assigned to a particular cluster. For 2 clusters the vulnerability score is the key feature that determines if a package is in the first or second cluster. For 3 clusters the vulnerability score and the community engagement score influence the cluster of a package. Also, it was found that the K-Means model produced more consistent clusters with less overlap when compared to the GMM model. Finally, it was found that the unweighted risk and reliability scores generated by the K-Means model have an inverse trend of data distribution when compared to the validation risk and reliability scores. Whereas the weighted risk and reliability scores have more packages with a higher risk score when compared to the unweighted scores. These weighted and unweighted risk scores are based on each package's distance to its cluster's center.

1.1.10 Discussion

Before discussing the results of the project, the results from the first half of the project must be discussed. The first half of the project attempted to cluster packages based on more than 20 features from the old dataset. This resulted in the below figures of clustering which contains many clusters. Due to the number of clusters being used in these figures being 56 this results in a clustering graph that has clusters that are not distinguished very well. This results in the model severely underperforming due to the large number of features used in the old dataset and the varying scales of all these features.

Figure 13: 1st Phase K-Means Clustering graph with K of 56 and random state of 59

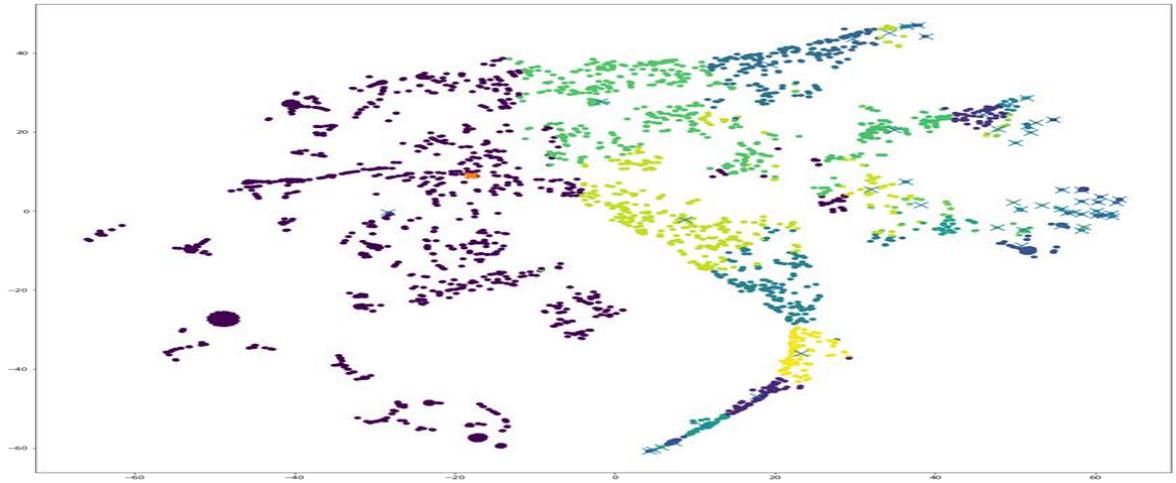
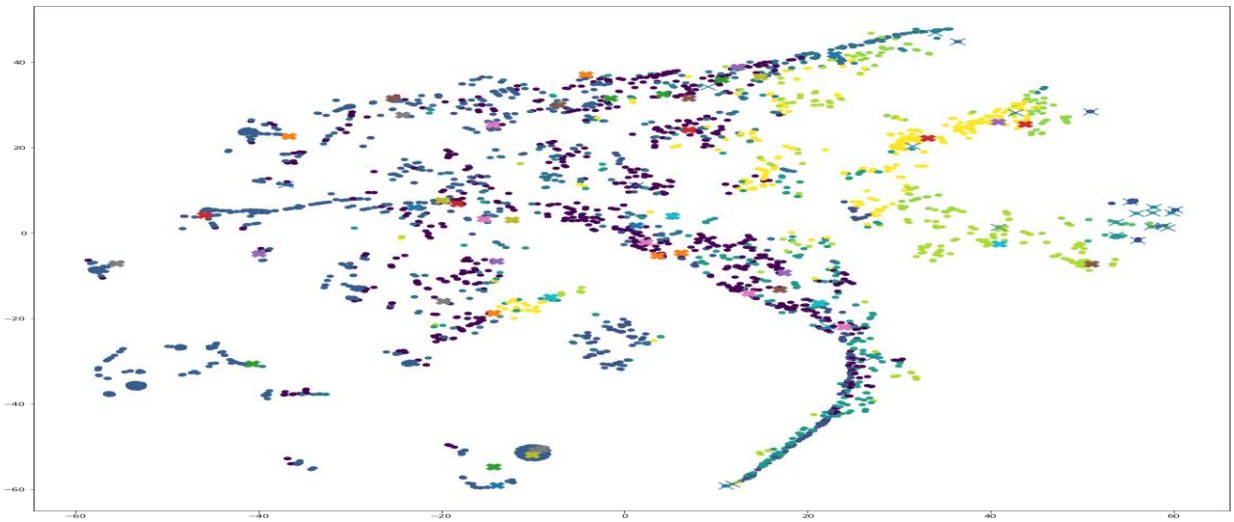


Figure 14: 1st Phase Gaussian Mixture Clustering Graph



Hence, for the second half of this project, a new dataset was used which consisted of 6 important features from the old dataset. In addition to this, the new dataset features are all in the same scale of a score out of 1 hence, the lack of standardization of the dataset in the

pre-processing stage for this project. This new dataset allows the clustering models to more reliably group clusters due to the features in the dataset being of the same scale and having fewer features in general.

The results of the clustering models show that both the K-Means and GMM models can group the packages within the dataset into distinct clusters. However, K-Means performs better when compared to the GMM model. This is shown in the decrease in overlap between the clusters in Figures 3 and 5.

The features within the dataset that contribute the most to the grouping of packages within a particular cluster are the vulnerability score and the community engagement score to a lesser extent. These features do have a considerable impact in the real world, in particular the vulnerability score. The vulnerability score is calculated based on the author of the package's email status and whether the email address is active or not. In the case of an author's email address being inactive, this gives a potential hacker the ability to buy the domain name to the email address and access the author's GitHub account and then make malicious changes to the package's source code.

The community engagement score was calculated based on the number of contributors a package has on GitHub. It is assumed that a reliable package will have more contributors. However, due to the open-source nature of the PyPi ecosystem, many programmers write the software for the package by themselves. Hence, just because a package has only one contributor does not mean it is malicious, however, it does make the package less reliable.

It was decided to use the K-Means model with 2 clusters and a random state of 59 for generating each package's distance to its cluster. 2 clusters were used under the assumption that in the real world, a package is either benign or malicious hence, the 2 clusters were used. The results have both the risk and reliability scores for 8000 packages. The risk score is the package's distance to its cluster where an increase in this score means that the package is further away from its cluster's center. Hence, the an increase in variability from the overall packages within that cluster. The calculation of the reliability score is a simple equation of subtracting the package's risk score from 1.

The data distribution for both scores is found in the histograms from Figures 6 and 7. The general trend observed for the risk score is that most packages have a risk score between 5 to 18%. Whereas most reliability scores are within the range of 80% to 94%. This is logically sound when compared to the real world. In the real world, it is expected that the majority of open-source PyPi packages are benign with no malicious intent and only a minority are malicious. However, this is not the case with validation reliability and risk scores where the general trend observed for the machine learning results is inverted.

As shown in Figures 11 and 12 most packages have a high risk score between 60% to 85% and for the reliability score, most packages have a score between 14% to 35%. This is a stark difference between the K-Means approach with cluster centres and the validation dataset. As

a result, it can be said that due to the way the score of risk and reliability are calculated this may be contributing to the large discrepancy in trends observed.

Another reason as to why this may be occurring could be due to the different characteristics of the two clusters found by the K-Means algorithm. It is discussed in Figure 4 that the main feature that separates clusters 1 and 2 from each other is the average vulnerability score. These average vulnerability scores are illustrated in the histogram found in Figure 4. Due to cluster 1 having a higher vulnerability score when compared to cluster 1, the generated risk scores from each package's distance to its cluster cannot be of the same scale as another cluster.

Hence, to fix this issue the risk scores for packages in cluster 1 were multiplied by 5 due to the average vulnerability score for cluster 1 being about 5 times higher than cluster 2. This resulted in a drastic change in the data distribution for the package's risk and reliability scores as illustrated in Figures 9 and 10. Now more packages have a higher risk score and fewer packages have a higher reliability score hence, aligning more with the data distribution found for the validation risk and reliability scores. However, this number that needs to be multiplied by the risk scores still needs to be refined.

1.1.11 Conclusion and Recommendations

As a result, it can be said that the project is a success. In terms of producing the initially set-out task of automated risk assessment of PyPi packages, the project is successful. However, the validation of this risk assessment has room for improvement, particularly with the difference in trends observed for the risk and reliability scores for the validation and clustering.

In the future, it would be recommended that the number of packages used for the clustering algorithms be increased. This is because a cluster algorithm’s performance increases as the number of data points increases. This allows for the algorithm to have a larger amount of data to train on and can result in the algorithm noticing more trends within a dataset.

1.1.12 References

Sejfia, A., & Schafer, M. (2022). Practical Automated Detection of Malicious npm Packages. 2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE), 1681–1692. https://doi.org/10.1145/3510003.3510104

Ohm, M., Boes, F., Bungartz, C., & Meier, M. (2022). On the Feasibility of Supervised Machine Learning for the Detection of Malicious Software Packages. ACM International Conference Proceeding Series. https://doi.org/10.1145/3538969.3544415

Kaplan, B., & Qian, J. (2021). A Survey on Common Threats in npm and PyPi Registries. https://doi.org/10.48550/arxiv.2108.09576

Bommarito, E., & Bommarito, M. (2019). An Empirical Analysis of the Python Package Index (PyPI). https://doi.org/10.48550/arxiv.1907.11073

Alfadel, M., Costa, D. E., & Shihab, E. (2023). Empirical analysis of security vulnerabilities in Python packages. Empirical Software Engineering : an International Journal, 28(3), 59–. https://doi.org/10.1007/s10664-022-10278-4

Liang, G., Zhou, X., Wang, Q., Du, Y., & Huang, C. (2021). Malicious Packages Lurking in User-Friendly Python Package Index. 2021 IEEE 20th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), 606–613. https://doi.org/10.1109/TrustCom53373.2021.00091

Heigl, M., Weigelt, E., Fiala, D., & Schramm, M. (2021). Unsupervised Feature Selection for Outlier Detection on Streaming Data to Enhance Network Security. *Applied Sciences*, *11*(24), 12073. https://doi.org/10.3390/app112412073

Executive Order on Improving the Nation's Cybersecurity 2021 (US)

Sanchez-Zas, C., Larriva-Novo, X., Villagra, V. A., Rodrigo, M. S., & Moreno, J. I. (2022). Design and Evaluation of Unsupervised Machine Learning Models for Anomaly Detection in Streaming Cybersecurity Logs. Mathematics (Basel), 10(21), 4043–. https://doi.org/10.3390/math10214043

Markonda Patnaikuni, H. (2023). Automated risk assessment of PyPi packages using unsupervised machine learning.

1.1.13 Appendix

Appendix 1 Research Project Timeline:

Number	Focus	Deliverable	Dependency	Release Date/ Milestone
1	Literature Review	Literature Review Report		21/04/2023
2	Conceptual design	Template of ipynb file (including loaded data and standardised data and general classification algorithm (e.g K Means)	1	05/05/2023
3	Sub project 1	Validating risk assessment with known malicious packages data set	1,2	26/05/2023
4	Interim Report	Report on key breakthroughs in research including programming and results aspects	1,2,3	26/05/2023
5	Sub project 2	Using another algorithm for generating risk assessment or using data analysis to analyse the data	1,2,3,4	21/07/2023
6	Detailed design	Pre-processing techniques used for data, multiple unsupervised machine learning algorithms to test, a criterion to determine what is considered risky, an end goal showing the output of the risk assessment	1,2,3,4,5	11/08/2023
7	Modeling/ Rapid prototype & experiment	ipynb file with pre-processing of data, unsupervised ML algorithms, a prototype risk assessment generator	1,2,3,4,5,6	12/09/2023
8	Interim Report	Report on key breakthroughs in research including programming and results aspects	1,2,3,4,5,6,7	29/09/2023
9	Pilot	Prototype of automated risk assessment tool	1,2,3,4,5,6,7,8	6/10/2023
10	Build/ Construct	ipynb file containing data pre-processing and algorithms	1,2,3,4,5,6,7,8,9	13/10/2023
11	Test & Validate/ Verify	Validate the ML algorithm with data about risk assessed packages and see if risk assessment is indicating that the package is risky.	1,2,3,4,5,6,7,8,9,10	20/10/2023
12	Draft report	Report on results of automated risk assessment	1,2,3,4,5,6,7,8,9,10,11	19/10/2023
13	Stakeholder consultation	Discussion with supervisor about final project	1,2,3,4,5,6,7,8,9,10,11,12	25/10/2023
14	Project Delivery Report & Oral Defence	Final algorithms, Python code, final report, final PowerPoint presentation	1,2,3,4,5,6,7,8,9,10,11,12,13,14	27/10/2023

Appendix 2 Python Code

egh400-1

October 24, 2023

1 CAB420, Practical 8 - Question 1 Solution

[1]:
```
import pandas
import numpy
from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture
from scipy.spatial import distance
import matplotlib.pyplot as plt
from sklearn.manifold import TSNE
from scipy.cluster.hierarchy import dendrogram
from matplotlib import cm
from datetime import datetime
```

1.1 Read the data and setup

[2]:
```
data = pandas.read_csv('Data20kcsv.csv');
#data['Author Email Status'] = data['Author Email Status'].map({'Success': 1,␣
↪'Bad': 0, 'SMTP Error': 0, "Absent":0, 'DNS Error':0})
print(data)
```

```
                       Package  SoftwareMaturityScore  FreshnessScore  \
0                        0-0-1               0.999916        1.000000
1                    0lever-so               0.999193        0.999918
2                0lever-utils               0.998450        0.999840
3      0x01-autocert-dns-aliyun               0.999937        1.000000
4                0x01-cubic-sdk               0.999911        0.999993
…                           …                      …               …
19994                  asserty               0.998008        0.999769
19995                  assess               0.999807        0.999976
19996                   asset               0.997128        0.999655
19997         asset-allocation               0.998262        0.999852
19998           assetallocator               0.999785        0.999942

       Accessibility_Score  Community_Engagement_Score  Vulnerability_Score  \
0                      0.2                    0.770533             0.460727
1                      0.4                    0.770533             0.114956
2                      0.4                    0.724129             0.460365
```

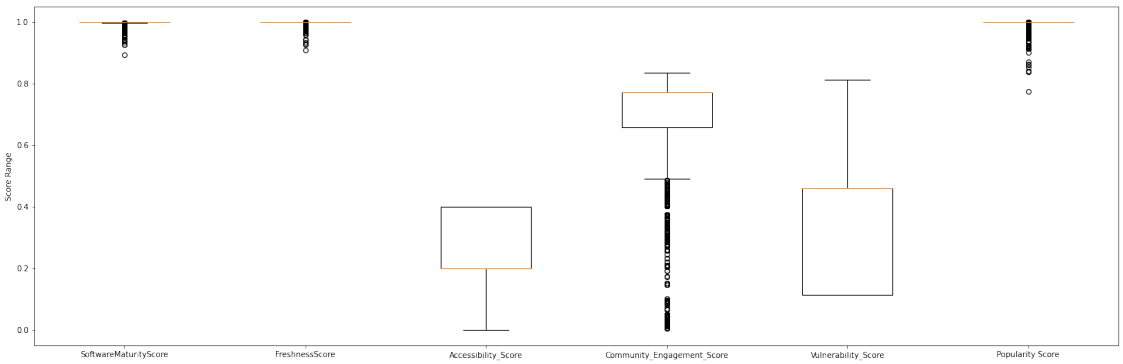
```
3                     0.2                    0.770533              0.460727
4                     0.2                    0.770533              0.460546
…                     …                       …                     …
19994                 0.4                    0.724129              0.460546
19995                 0.4                    0.770533              0.461812
19996                 0.2                    0.724129              0.466876
19997                 0.2                    0.770533              0.122189
19998                 0.2                    0.724129              0.114956

       Popularity Score  FinalRiskScore
0              0.999986        0.831905
1              0.999984        0.710778
2              0.999997        0.836168
3              1.000000        0.831910
4              0.999997        0.831817
…                 …             …
19994          0.999997        0.836134
19995          0.999997        0.848123
19996          0.999997        0.822576
19997          0.999986        0.697521
19998          0.999995        0.684086

[19999 rows x 8 columns]
```

[3]:
```
#train = data.iloc[0:3000, [2,3,8,12,13,14,15,16,17,18,19]]
#test = data.iloc[3000:6000, [2,3,8,12,13,14,15,16,17,18,19]]

#train = data.iloc[0:3000, [2,3,8,12]]
#test = data.iloc[3000:6000, [2,3,8,12]]

#train = data.iloc[0:3000, [2,3,8,12,13,14,15]]
#test = data.iloc[3000:6000, [2,3,8,12,13,14,15]]

# Features to consider from new dataset

#SoftwareMaturityScore
#FreshnessScore
#Accessibility_Score
#Community_Engagement_Score
#Vulnerability_Score
#Popularity Score
#FinalRiskScore

#train = data.iloc[0:3000, [2,3,8,12,13,14,15,16,17,18,19,35]]
#test = data.iloc[3000:6000, [2,3,8,12,13,14,15,16,17,18,19,35]]

#train = data.iloc[0:50000, [2,3,8,12,13,14,15,16,17,18,19,35,38]]
```

```
#test = data.iloc[50000:100000, [2,3,8,12,13,14,15,16,17,18,19,35,38]]

#train = data.iloc[0:50000, [2,3,8]]
#test = data.iloc[50000:100000, [2,3,8]]

#train = data.iloc[0:8000, [1,2,3,4,5,6,7]]
#test = data.iloc[8000:16000, [1,2,3,4,5,6,7]]

train = data.iloc[0:8000, [1,2,3,4,5,6]]
test = data.iloc[8000:16000, [1,2,3,4,5,6]]

#train = data.iloc[0:8000, [3,4,5]]
#test = data.iloc[8000:16000, [3,4,5]]

#mu = numpy.mean(train)
#sigma = numpy.std(train)

#train = (train - mu) / sigma;
#test = (test - mu) / sigma;
```

[7]:
```
fig = plt.figure(figsize=[25, 8])
ax = fig.add_subplot(1, 1, 1)
xlabels = ['Software Maturity Score', 'Freshness Score', 'Accessibility Score',␣
 ↪'Community Engagement Score', 'Vulnerability Score', 'Popularity Score']
ylabel = 'Score Range'
ax.boxplot(train)
ax.set_xticklabels(labels)
ax.set_ylabel(ylabel)
```

[7]:
```
Text(0, 0.5, 'Score Range')
```



2 K-Means Algorithm

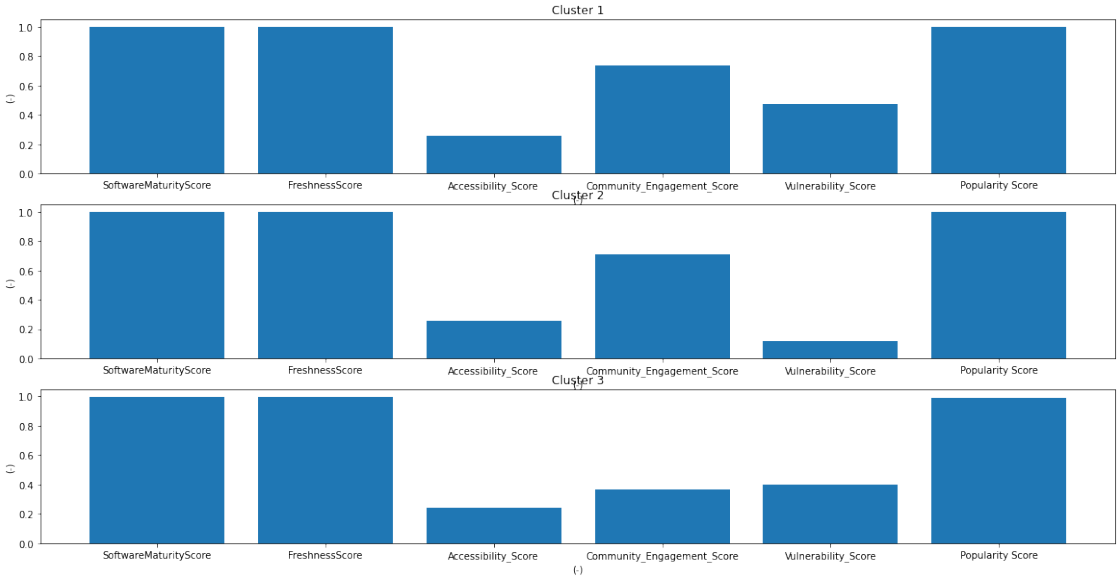
[16]:
```
#
# K-Means Analysis from CAB420, Prac 8, Q1
# Author: Simon Denman (s.denman@qut.edu.au)
#
def do_kmeans_analysis(n_clusters, random_state, train, test,␣
 ↪abnormal_threshold = 0.7):

    # train the k-means model
    kmeans = KMeans(n_clusters=n_clusters, random_state=random_state,␣
 ↪n_init='auto').fit(train)
    cluster_labels = kmeans.predict(test)

    # plot the cluster centres
    #x = ['Contributors', 'No_Of_Releases', 'Mean_Distance', 'Number of Stars',␣
 ↪'Closed_Issues', 'Open_Issues', 'Total_Branches', 'Open_Pull_Requests',␣
 ↪'Closed_Pull_Requests','Open_Milestones','Closed_Milestones']
    #x = ['Contributors', 'No_Of_Releases', 'Mean_Distance', 'Number of Stars']
    #x = ['Contributors', 'No_Of_Releases', 'Mean_Distance', 'Number of Stars',␣
 ↪'Closed_Issues', 'Open_Issues', 'Total_Branches']
    #x = ['Contributors', 'No_Of_Releases', 'Mean_Distance', 'Number of Stars',␣
 ↪'Closed_Issues', 'Open_Issues', 'Total_Branches', 'Open_Pull_Requests',␣
 ↪'Closed_Pull_Requests','Open_Milestones','Closed_Milestones', 'Avg Package␣
 ↪Size']
    #x = ['Contributors', 'No_Of_Releases', 'Mean_Distance', 'Number of Stars',␣
 ↪'Closed_Issues', 'Open_Issues', 'Total_Branches', 'Open_Pull_Requests',␣
 ↪'Closed_Pull_Requests','Open_Milestones','Closed_Milestones', 'Avg Package␣
 ↪Size', 'Author Email Status']
    #x = ['Contributors', 'No_Of_Releases', 'Mean_Distance']
    #x = ['SoftwareMaturityScore', 'FreshnessScore', 'Accessibility_Score',␣
 ↪'Community_Engagement_Score', 'Vulnerability_Score', 'Popularity Score',␣
 ↪'FinalRiskScore']
    x = ['SoftwareMaturityScore', 'FreshnessScore', 'Accessibility_Score',␣
 ↪'Community_Engagement_Score', 'Vulnerability_Score', 'Popularity Score']
    #x = ['Accessibility_Score', 'Community_Engagement_Score',␣
 ↪'Vulnerability_Score']
    fig = plt.figure(figsize=[20, 10])
    for i in range(n_clusters):
        ax = fig.add_subplot(n_clusters, 1, i + 1)
        ax.bar(x, kmeans.cluster_centers_[i,:])
        ax.set_title('Cluster %d' % (i+1))
        ax.set_xlabel('(-)')
        ax.set_ylabel('(-)')

    # plot the data, we can't plot 10-d data, so we'll run TSNE and show that
```

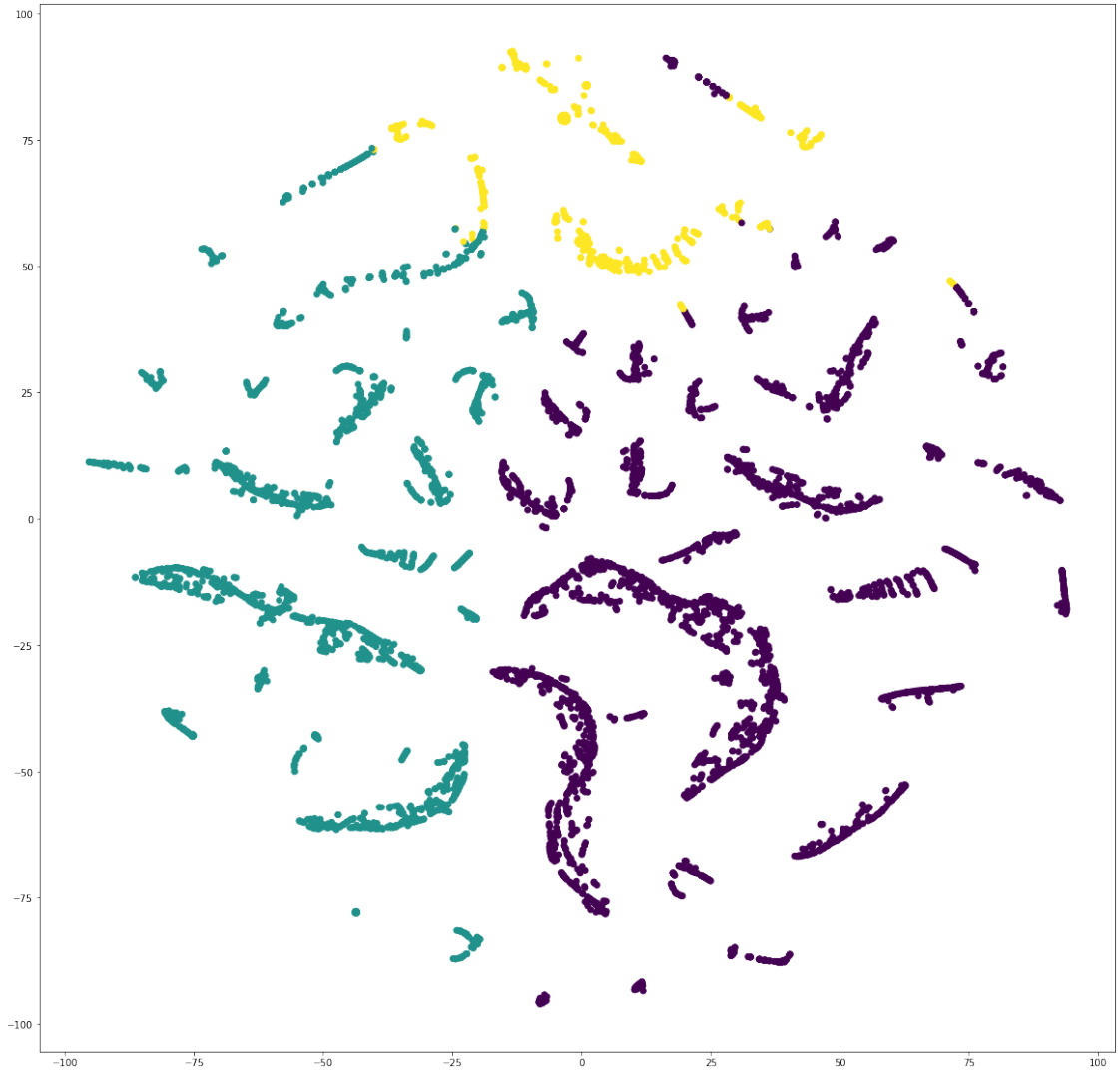
```
    fig = plt.figure(figsize=[20,20])
    ax = fig.add_subplot(1, 1, 1)
    tsne_embeddings = TSNE(random_state=4).fit_transform(numpy.vstack([train,␣
↪kmeans.cluster_centers_]))
    ax.scatter(tsne_embeddings[:-n_clusters,0], tsne_embeddings[:
↪-n_clusters,1], c = kmeans.labels_);
    #add cluster centres to the plot
    ax.scatter(tsne_embeddings[-n_clusters:,0], tsne_embeddings[-n_clusters:
↪,1], s=200, marker='x')

    # find abnormal travellers. With k-means, we'll use distance to the cluster␣
↪centre as our measure
    distances = kmeans.transform(test)
    distances = numpy.min(distances, axis=1)


    print('Risk Scores:')
    for count, (dist, cluster) in enumerate(zip(distances, cluster_labels)):
            #print(f'{cluster}')
        if cluster == 0:
        # If the package is in Cluster 0, multiply the distance by 5
            dist *= 5
        # If the new distance is greater than 1, set it to 1
            if dist > 1:
                dist = 1

        print(f'{dist}')

    print('Reliability Scores:')
    for count, (dist, cluster) in enumerate(zip(distances, cluster_labels)):
            #print(f'{cluster}')
        if cluster == 0:
        # If the package is in Cluster 0, multiply the distance by 5
            dist *= 5
        # If the new distance is greater than 1, set it to 1
            if dist > 1:
                dist = 1

        print(f'{1 - dist}')
```

We'll run three versions of K-means, with different random seeds.

[ ]:
```
do_kmeans_analysis(2, 59, train, test)
```

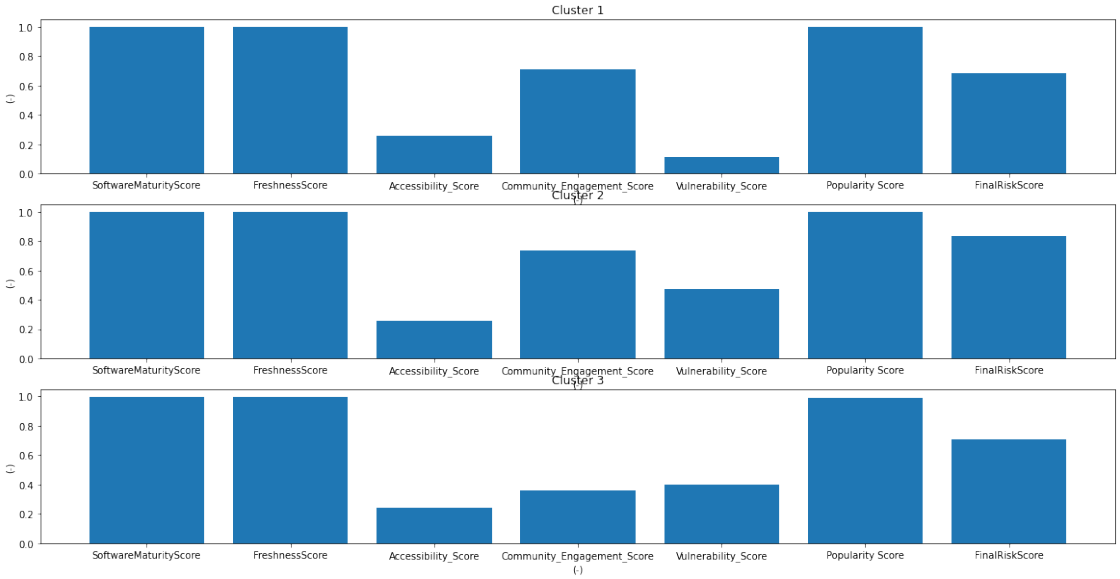
[11]:
```
do_kmeans_analysis(3, 314, train, test)
```

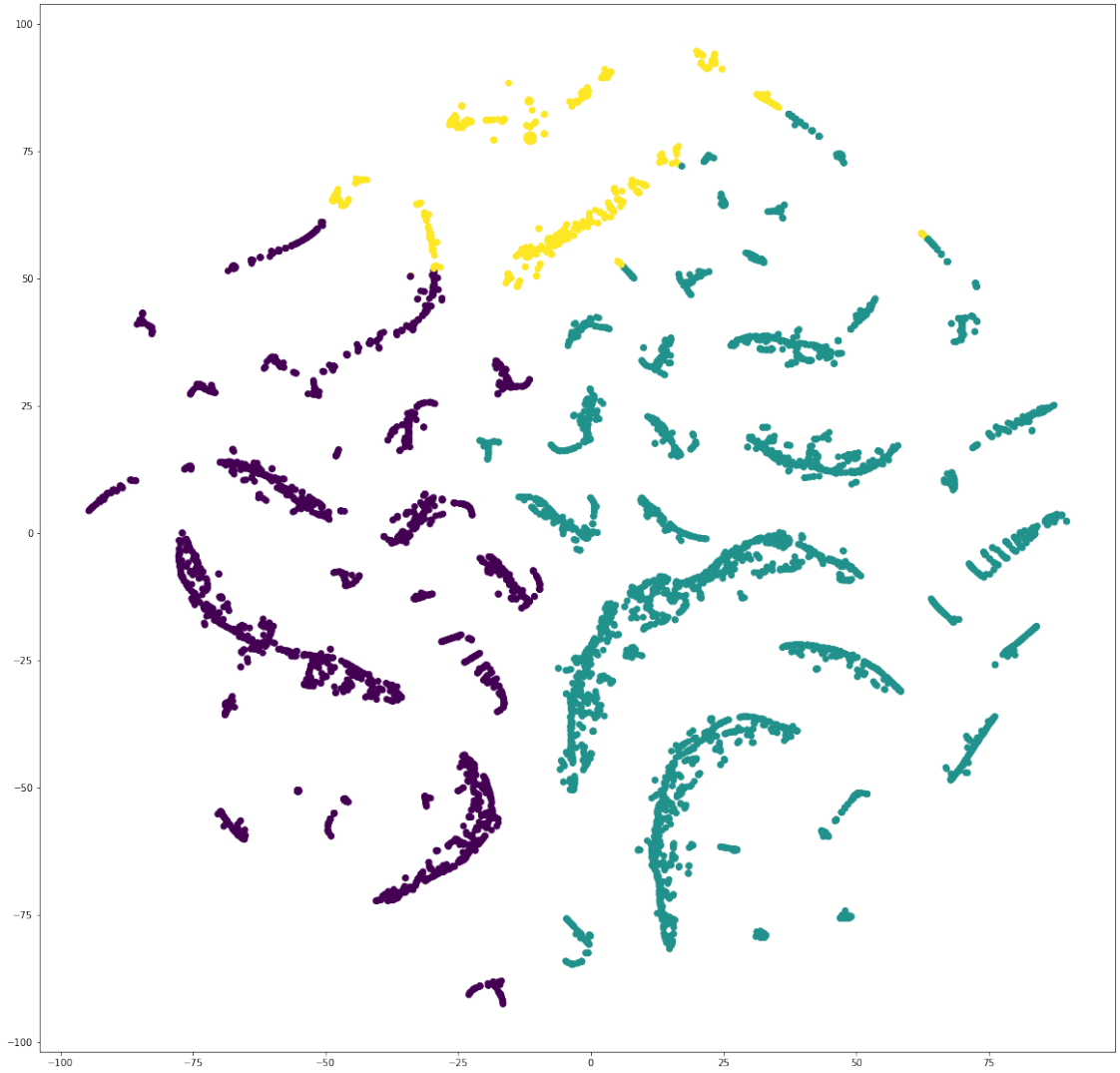

Cluster 1
(-)
1.0
0.8
0.6
0.4
0.2
0.0
SoftwareMaturityScore
FreshnessScore
Accessibility_Score
Community_Engagement_Score
Vulnerability_Score
Popularity Score
(-)
Cluster 2
(-)
1.0
0.8
0.6
0.4
0.2
0.0
SoftwareMaturityScore
FreshnessScore
Accessibility_Score
Community_Engagement_Score
Vulnerability_Score
Popularity Score
(-)
Cluster 3
(-)
1.0
0.8
0.6
0.4
0.2
0.0
SoftwareMaturityScore
FreshnessScore
Accessibility_Score
Community_Engagement_Score
Vulnerability_Score
Popularity Score
(-)



<Figure size 1800x1440 with 0 Axes>

[30]:
```
do_kmeans_analysis(3, 200, train, test)
```


Cluster 1
Cluster 2
Cluster 3
(-)
1.0
0.8
0.6
0.4
0.2
0.0
SoftwareMaturityScore
FreshnessScore
Accessibility_Score
Community_Engagement_Score
Vulnerability_Score
Popularity Score
FinalRiskScore



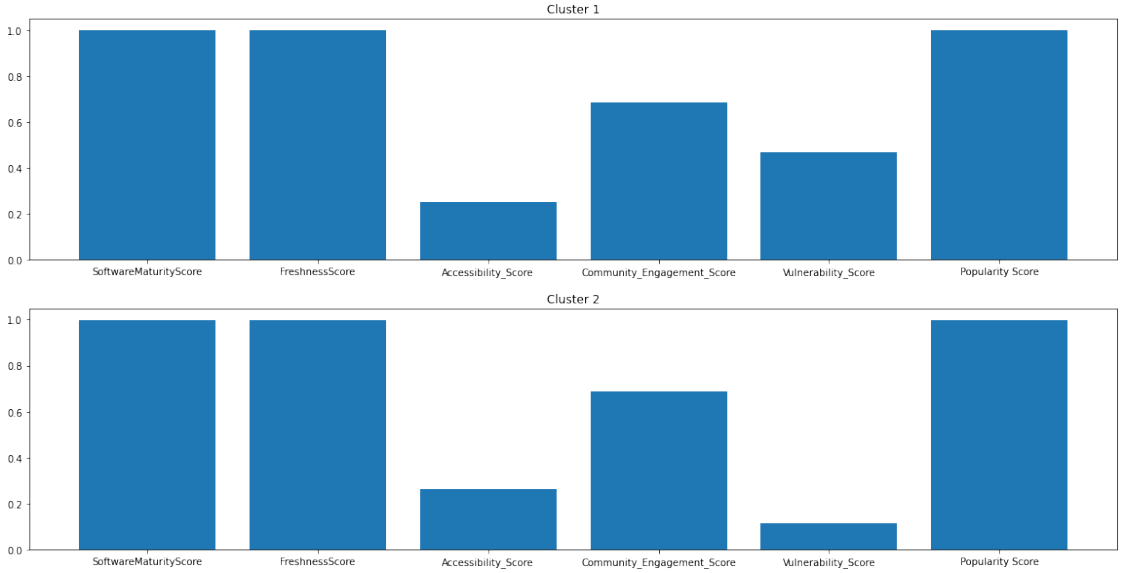
<Figure size 1800x1440 with 0 Axes>

2.0.1 GMM

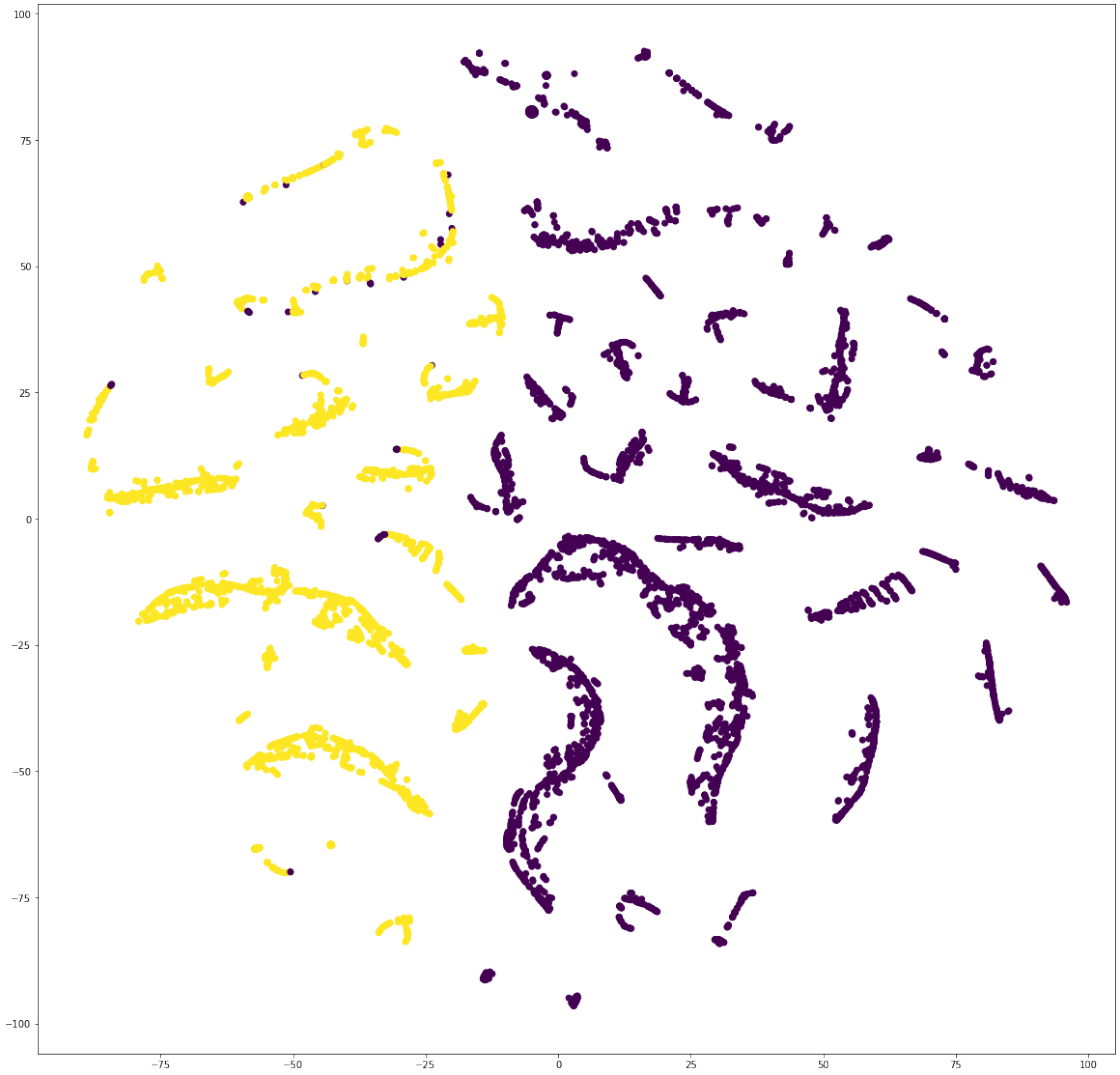
[18]:
```
#
# GMM Analysis from CAB420, Prac 8, Q1
# Author: Simon Denman (s.denman@qut.edu.au)
#
def do_gmm_analysis(n_components, random_state, train, test):

    # train the GMM
    gmm = GaussianMixture(n_components=n_components, random_state=random_state).
 ↪fit(train)
```

```
    # plot the component means
    #x = ['Contributors', 'No_Of_Releases', 'Mean_Distance', 'Number of Stars',␣
↪'Closed_Issues', 'Open_Issues', 'Total_Branches', 'Open_Pull_Requests',␣
↪'Closed_Pull_Requests','Open_Milestones','Closed_Milestones']
    x = ['SoftwareMaturityScore', 'FreshnessScore', 'Accessibility_Score',␣
↪'Community_Engagement_Score', 'Vulnerability_Score', 'Popularity Score']
    fig = plt.figure(figsize=[20, 10])
    for i in range(n_components):
        ax = fig.add_subplot(n_components, 1, i + 1)
        ax.bar(x, gmm.means_[i,:])
        ax.set_title('Cluster %d' % (i+1))

    # TSNE plot of the data
    fig = plt.figure(figsize=[20,20])
    ax = fig.add_subplot(1, 1, 1)
    tsne_embeddings = TSNE(random_state=4).fit_transform(numpy.vstack([train,␣
↪gmm.means_]))
    train_labels = gmm.predict(train)
    ax.scatter(tsne_embeddings[:-n_components,0], tsne_embeddings[:
↪-n_components,1], c = train_labels);

    # add component means to the scatter plot
    #ax.scatter(tsne_embeddings[-n_components:,0],␣
↪tsne_embeddings[-n_components:,1], s=200, marker='x')

    # compute likelihood for travellers. With a GMM, we can get the likelihood␣
↪that such a traveller exists
    # given the learned distribution
    distances = gmm.score_samples(test)
```

[19]:
```
do_gmm_analysis(2, 59, train, test)
```


Cluster 1
1.0
0.8
0.6
0.4
0.2
0.0
SoftwareMaturityScore
FreshnessScore
Accessibility_Score
Community_Engagement_Score
Vulnerability_Score
Popularity Score

Cluster 2
1.0
0.8
0.6
0.4
0.2
0.0
SoftwareMaturityScore
FreshnessScore
Accessibility_Score
Community_Engagement_Score
Vulnerability_Score
Popularity Score


100
75
50
25
0
−25
−50
−75
−100
−75
−50
−25
0
25
50
75
100

[ ]: