Points Noted:

-Half portion is padded. Many medical images are stored in square matrices (e.g., 512x512) even if the anatomy occupies only a portion.

* Threshold based methods, simply plot a histogram and consider peaks as classes
* Initial intuition: Seems like binary segmentation task, there are no complex structures so Threshold might work,
* Edge detection methods(canny edge detection and applying morphological techniques
* A voxel, short for "volume pixel," is the 3D equivalent of a pixel, representing a value in a discretized 3D space. In medical imaging, voxels correspond to volumetric elements in MRI or CT scans. By stacking 2D slices along a third, depth dimension, we obtain a volumetric dataset composed of these voxels
* ITK has a number of histogram based automatic thresholding filters including Huang, MaximumEntropy, Triangle, and the popular Otsu's method. These methods create a histogram then use a heuristic to determine a threshold value.
* Region growing is another popular segmentation technique, which involves starting from a seed point and progressively adding neighboring pixels to the region if they meet certain criteria. The ConnectedThreshold() function in SimpleITK allows you to perform region growing:

Task 1.2

-To uniformly expand a segmented mask outward, you can use techniques like dilation or expansion in image processing. This involves adding a "halo" or "border" around the existing mask, effectively increasing its area while maintaining the original shape.

-dilation must expand the **contours by a physical distance** (e.g., 2 mm), not by just a fixed number of voxels.

-To do this correctly:

* Convert **2 mm** to **voxel units** using image spacing.
* Apply **dilation** with a ball-shaped kernel whose radius matches the desired 2 mm expansion.
* Apply it separately to **femur (label 1)** and **tibia (label 2)**.
* Keep the **labels** intact in the output mask.

Task 1.3

-This task is essentially **morphological contour jittering** — adding **random variation** in contour, but only **within a valid band**:

-Randomized contour adjustment usually means modifying organ or tissue contours using random perturbations to:

* Simulate anatomical variability
* Improve model robustness
* Perform uncertainty modeling

-Use Google Scholar or PubMed to search for medical imaging data augmentation or shape transforms papers

-MONAI

Task 1.4

# Results

Original:

Medial = (np.float64(9.1552734375e-05), np.float64(127.76381599903107), np.float64(-470.5))

Lateral = (np.float64(197.29509460926056), np.float64(88.65247178077698), np.float64(-470.5))

Expanded 2mm:

Medial = (np.float64(-2.607331395149231), np.float64(126.8946750164032), np.float64(-470.5))

Lateral = (np.float64(199.90251755714417), np.float64(87.78333079814911), np.float64(-470.5))

Expanded 4mm:

Medial = (np.float64(-2.607331395149231), np.float64(126.8946750164032), np.float64(-470.5))

Lateral = (np.float64(199.90251755714417), np.float64(87.78333079814911), np.float64(-470.5))

Randomized 1:

Medial = (np.float64(-2.607331395149231), np.float64(126.8946750164032), np.float64(-470.5))

Lateral = (np.float64(199.90251755714417), np.float64(87.78333079814911), np.float64(-470.5))

Randomized 2:

Medial = (np.float64(-2.607331395149231), np.float64(127.76381599903107), np.float64(-470.5))

Lateral = (np.float64(199.0333765745163), np.float64(90.39075374603271), np.float64(-470.5))

#Task2

-Data preprocessing (remove duplicates, scaling,

-EDA(outlier, viz, uni, bi, imbalance)

-Feature engineering and selection

??? Handling missing data

-The V columns appear to be redundant and correlated. Therefore for each block of V columns with similar NAN structure, we could find subsets within the block that are correlated (r > 0.75). Then we can replace the entire block with one column from each subset.

-For example in block V1-V11, we see that the subsets [[1],[2,3],[4,5],[6,7],[8,9],[10,11]] exist and we can choose [1, 3, 4, 6, 8, 11] to represent the V1-V11 block without losing that much information. Alternatively you could apply PCA on each block, but this subset reduce method performed better.

-It can be surprising that **more features might be equal to worse performance**. There's many possible reasons, but one might be that those features are too noisy and cause the model to **over-fit**: fit specific training data-points instead of generalizing to be usable on any data-point.

**Random Forest or XGBoost** – They can handle high dimensions well and give feature importances.

➡️ **Support Vector Machines (SVM)** – Works well in high-dimensional spaces, though it may need tuning.

✅ **Model-based selection** – Use a simple model (like Random Forest or Lasso) to rank feature importances

✅ **Variance Thresholding** – Remove features with very low variance (they’re nearly constant)

⚙️ **1. Dimensionality Reduction (PCA- loses some features)**

✅ **LDA (Linear Discriminant Analysis)** – Good for binary classification; tries to find the best feature projection for separating classes.

Using 'roc\_auc' for **imbalanced datasets** is better than 'accuracy'.

-Recursive feature elimination (RFE) is a feature selection technique that helps us to select best features from the given number of features. At first, the model is built on all the given features. Then, it removes the least useful predictor and build the model again. This process is repeated until all the unimportant features are removed from the model.

Recursive Feature Elimination with Cross-Validated (RFECV) feature selection technique selects the best subset of features for the estimator by removing 0 to N features iteratively using recursive feature elimination. Then it selects the best subset based on the accuracy or cross-validation score or roc-auc of the model. Recursive feature elimination technique eliminates n features from a model by fitting the model multiple times and at each step, removing the weakest features.

I will use this technique to select best features from this model.

In [145]:

linkcode

from sklearn.feature\_selection import RFECV

rfecv = RFECV(estimator=logreg, step=1, cv=5, scoring='accuracy')

rfecv = rfecv.fit(X\_train, y\_train)

2. Check Columns with Similar Values

Now, we find out the features where the majority share of the values are similar. These features will not help us differentiate between customers and hence we drop the features where more than 90% of values are similar.

|  |  |
| --- | --- |
|  | def get\_similar\_value\_cols(df, percent=90): |
|  | """ |
|  | :param df: input data in the form of a dataframe |
|  | :param percent: integer value for the threshold for finding similar values in columns |
|  | :return: sim\_val\_cols: list of columns where a singular value occurs more than the threshold |
|  | """ |
|  | count = 0 |
|  | sim\_val\_cols = [] |
|  | for col in df.columns: |
|  | percent\_vals = (df[col].value\_counts()/len(df)\*100).values |
|  | # filter columns where more than 90% values are same and leave out binary encoded columns |
|  | if percent\_vals[0] > percent and len(percent\_vals) > 2: |
|  | sim\_val\_cols.append(col) |
|  | count += 1 |
|  | print("Total columns with majority singular value shares: ", count) |
|  | return sim\_val\_cols |

<https://spotintelligence.com/2024/11/14/handling-high-dimensional-data/>

* There are categorical data with numerical values,
* Most features are unique
* ID is object type, other are float64 with target column int64
* 114 columns with 1 unique value which contribute nothing in output so should be removed
* There are more than 100 categorical column so one hot encoding them is not feasible, recommended to use tree based methods which handles numerical values of categorical data
* Tree-based models (like Random Forests, Gradient Boosting) are generally not sensitive to feature scaling
* Linear models, SVMs, and neural networks *are* sensitive to feature scaling.

**Feature Selection / Dimensionality Reduction (Crucial for 3500 Columns):**

* Even after cleaning, 3500 features is a lot for many ML algorithms and can lead to overfitting, long training times, and difficulty interpreting results.
* **Methods:**
  + **Filter Methods:** Select features based on statistical measures (e.g., correlation with target, chi-squared test for categorical features, variance threshold). Use sklearn.feature\_selection modules like SelectKBest or VarianceThreshold.
  + **Wrapper Methods:** Select features by training a model and evaluating feature subsets (e.g., Recursive Feature Elimination - RFE). Computationally expensive.
  + **Embedded Methods:** Feature selection is built into the model training process (e.g., L1 regularization in linear models, feature importance in tree-based models).
  + **Dimensionality Reduction:** Techniques like Principal Component Analysis (PCA) or t-SNE (primarily for visualization) can create a smaller set of new features. PCA is suitable for numerical data.
* **Implementation using Pipelines:** Use a ColumnTransformer to apply scaling specifically to your numerical\_cols.

numerical\_transformer = Pipeline(steps=[

('imputer', SimpleImputer(strategy='mean')), # Or 'median'

('scaler', StandardScaler()) # Or MinMaxScaler()

])

preprocessor = ColumnTransformer(

transformers=[

('num', numerical\_transformer, numerical\_cols),

# Add other transformers for categorical if needed

# ('cat', categorical\_transformer, categorical\_cols)

],

# Set remainder='passthrough' to keep columns not listed, or 'drop' to remove them

# If your categorical columns are just numerical values you want to keep, use 'passthrough'

# or include them in a separate transformer if they need imputation but not scaling/encoding.

remainder='passthrough' # Keep the categorical\_cols as they are after numerical processing

)

* There are inf values present in dataset
* Logistic Regression is heavily effected by outliers
* If data is normally distributed we can detect outlier using z score method(out of mu+3sigma,mu-3sigma) are outlier
* If data is skewed we can use box plot
* Categorical data can be oridinal and nominal.
* Ordinal: some meaning in each category
* Nominal: no relation among categories
* Logistic Regression is effected by log transform but DT are not
* There are missing values in test dataset
* When performing Label Encoding below, you must encode train and test together as in
* df = pd.concat([train[col],test[col]],axis=0)
* # PERFORM FEATURE ENGINEERING HERE
* train[col] = df[:len(train)]
* test[col] = df[len(train):]
* Features are heavinly correlated
* DecisionTreeRegressor
* X\_log= X.applymap(lambda x: np.log(x) if isinstance(x, (int, float)) **and** x > 0 else x)
* *# If you want to apply the transformation only to specific columns, you can specify them like this:*
* *# df\_log = df[['column1', 'column2', ...]].applymap(lambda x: np.log(x) if isinstance(x, (int, float)) and x > 0 else x)*
* X\_log.head()
* X\_log["Index"]=df["Index"]
* Look for negative values
* Applying IQR method to hanlde missing values
* import numpy as np
* from scipy.stats import chi2\_contingency
* # Recalculate the Chi-Square test after grouping or resampling
* crosstab = pd.crosstab(df\_train['loan\_status'], df\_train['grade'])
* def cramers\_v(chi2, n, k1, k2):
* return np.sqrt(chi2 / (n \* (min(k1, k2) - 1)))
* chi2\_stat, p\_val, dof, expected = chi2\_contingency(crosstab)
* cramers\_v\_value = cramers\_v(chi2\_stat, len(df\_train), len(crosstab.columns), len(crosstab.index))
* print("Cramér's V:", cramers\_v\_value,p\_val)

Apply corr matrix to log transform

Pca analysis

Apply K-means, DBSCAN, ensemble techniques

Handle negative values

from sklearn.experimental import enable\_iterative\_imputer # noqa

from sklearn.impute import IterativeImputer

# Assuming X is your feature matrix with missing values

# Create IterativeImputer instance

imputer = IterativeImputer(random\_state=42)

# Fit the imputer to the data and transform it to impute missing values

X\_imputed = imputer.fit\_transform(X\_log.iloc[:,1:])

X\_tested = imputer.fit\_transform(X\_test2.iloc[:,:27])

ximp\_df=pd.DataFrame(X\_imputed)

ximptest\_df = pd.DataFrame(X\_tested)

ximptest\_df

Model is overfitting

imputer = IterativeImputer(random\_state=42)

# Fit the imputer to the data and transform it to impute missing values

X\_imputed = imputer.fit\_transform(test\_df\_log)

test\_df=pd.DataFrame(X\_imputed,columns=columns)

In same column in train, test, and blind test there are missing values