Niketan Baranwal

Missing Data Assignment

The dataset used for this assignment is the infamous Titanic dataset which is used to teach many students worldwide about missing values in a dataset.

The goal of this assignment is to try and predict the survivability rate of the passengers dependent on different features in the dataset such as sex, age, passenger class, etc…

We explore different ways of handling missing values in datasets using 3 types of classifiers:

1. **Linear Regression Models**
2. **Tree Models**
3. **Neural Networks**

1. **Linear Regression Models**

```
import pandas as pd
import numpy as np
```

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.impute import SimpleImputer
```

We import multiple packages to work with the right tools that can help us seamlessly create a linear regression model.

The Pandas and Numpy packages are meant to be used for data manipulation and handling, meanwhile, the various packages from sklearn are imported to help us create the linear regression model.

```
titanic_data = pd.read_csv("/Users/niketanbaranwal/Downloads/Titanic-Dataset.csv")
```

```
#get a glimpse of the dataset that we are working with and what types of categorical and numerical attributes exist
titanic_data.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

```
#helps us see where most of the data is missing from: age, cabin, embarked
titanic_data.isnull().sum()
```

```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

We import the dataset from our local device into Python and immediately check to see if all the attributes and features have been imported correctly by using .head(). Our main goal is to check how much missing data already exists in the dataset so we can understand what features need to be imputed to have no missing values.

```
titanic_data.dtypes
```

```
PassengerId      int64
Survived         int64
Pclass           int64
Name            object
Sex             object
Age            float64
SibSp            int64
Parch            int64
Ticket          object
Fare           float64
Cabin           object
Embarked        object
dtype: object
```

```
#numerical colums are ['PassengerId','Survived','Pclass','Age','SibSp','Parch','Fare']
```

```
#categorical columns are ['Name', 'Sex', 'Ticket', 'Cabin','Embarked']
```

We check the data types of the attributes to see which values are numerical and which are categorical. This helps us decide which type of imputation we can use on which attribute.

```
#drop cabin column, so many missing values can make imputation difficult plus imputing such a large number of data can cause bias
titanic_data = titanic_data.drop('Cabin',axis=1)
```

```
#median imputation since age is a numerical attribute
imputer_age = SimpleImputer(strategy = 'median')
titanic_data['Age'] = imputer_age.fit_transform(titanic_data[['Age']])
```

```
#mode imputation since embarked is a catergorical attribute
imputer_embarked = SimpleImputer(strategy='most_frequent')
titanic_data['Embarked'] = imputer_embarked.fit_transform(titanic_data[['Embarked']])[:,0]
```

We drop the cabin column because too many missing values exist to perform imputation on that attribute. But even if we did decide to impute the cabin attribute, it would end up causing bias due to the imputation of many missing values which can make it appear more influential than it is.

We can then perform median imputation on the age attribute. Median imputation is chosen in this scenario because age distributions can vary significantly meaning they might be skewed or have multiple peaks, the mean imputation can flatten these distributions causing them to look more normal than they are.

We perform a mode imputation on the embarked attribute since it is a categorical variable. Mean and median imputations have no real effect on categorical variables.

```
#one hot encode categorical features: changes categorical values into numerical values (0 or 1) and reduces redundancy/avoids multicollinearit
titanic_data = pd.get_dummies(titanic_data,columns = ['Sex','Embarked'], drop_first = True)
```

```
#drop unnecessary columns since we are trying to predict survivability, these columns do not matter to us
titanic_data = titanic_data.drop(['PassengerId', 'Name' ,'Ticket'], axis = 1)
```

```
X = titanic_data.drop('Survived',axis = 1)
y = titanic_data['Survived']
```

We one-hot encode categorical features because ML algorithms cannot read categorical data, they need numerical data. This also helps us reduce redundancy and avoid collinearity. For example, one hot encoding the “sex” attribute will create 2 columns such as “Sex: Female” and “Sex: Male”. The “Sex: Female” column will have a 1 where the “Sex” attribute is female and a 0 where the “Sex” attribute is male. The “Sex: Male” attribute will have a 1 where the “Sex” attribute is male and a 0 where the “Sex” attribute is female. This is redundant as both columns give us the same information so we can have the drop_first condition and set it to True which drops one of the redundant columns.

We can then also drop unnecessary columns that we understand will not help us determine the survivability of the passengers on the Titanic.

Now we create two different data frames, X which does not contain the “Survived” attribute, and y which contains only the “Survived” attribute.

```
titanic_data.head()
```

	Survived	Pclass	Age	SibSp	Parch	Fare	Sex_male	Embarked_Q	Embarked_S
0	0	3	22.0	1	0	7.2500	True	False	True
1	1	1	38.0	1	0	71.2833	False	False	False
2	1	3	26.0	0	0	7.9250	False	False	True
3	1	1	35.0	1	0	53.1000	False	False	True
4	0	3	35.0	0	0	8.0500	True	False	True

```
titanic_data.isnull().sum()
```

```
Survived      0
Pclass        0
Age           0
SibSp         0
Parch         0
Fare          0
Sex_male      0
Embarked_Q    0
Embarked_S    0
dtype: int64
```

We now double-check if the data has everything we specified that we wanted and if it doesn’t have attributes that we no longer need.

We re-run the .isnull().sum() function from before to check if we have null values left in our dataset. In the example above, all the null values were eliminated by statistical imputations.

```
scaler = StandardScaler()
numerical_cols = ['Pclass','Age','SibSp','Parch','Fare']
X[numerical_cols] = scaler.fit_transform(X[numerical_cols])
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=50)
```

```
#
linear_model = LogisticRegression(max_iter=1000)
linear_model.fit(X_train,y_train)
y_pred_linear = linear_model.predict(X_test)
linear_accuracy = accuracy_score(y_test, y_pred_linear)
print(f"Linear Model Accuracy: {linear_accuracy}")
```

```
Linear Model Accuracy: 0.8044692737430168
```

We apply standardization to the numerical columns in our data. We first define the numerical_cols and put it through StandardScaler using the fit_transform function. This first calculates the mean and standard deviation for each column from the data in X (done once per column). We then apply standardization transformation by transforming the mean to 0 and the standard deviation to 1. After standardization, some data points will have positive values while some will have negative values which balance each other out. The standard deviation generally measures the dispersion of the data so a standard deviation of 1 means that the data points, on average, are one point away from the mean. This helps to prevent features with larger ranges from dominating the model.

We set the features for the training set in X_train and the features in the test set in X_test. We also set the target training variable in y_train and the target testing variable in y_test. We set test_size equal to .2 meaning that 20% of the data will be used to test while the other 80% will be used to train the model.

The random_state parameter in machine learning functions is used to control the randomness of the process. It's a seed for a random number generator. We set that value equal to 50. Setting random_state to a specific integer (like 50) ensures that the data is split the same way every time you run the code. This is crucial for reproducibility.

We can finally create the Liner Regression Model which, in this case, would be a Logistic Regression Model. Since the relationship between the features and the probability of survival is non-linear, we can use a Logistic Regression model (“sigmoid” function) to transform the linear combination of the features into a probability between 0 to 1. If we used Linear Regression to predict “Survived”, it would try to fit a line to predict values that could be anywhere from negative infinity to positive infinity. Those results could be rounded to 0 or 1 but would not be optimized for this type of problem.

Ultimately, we achieved an 80.4% accuracy score which is reasonable given that the Titanic dataset is a complex problem.

2. **Tree Models**

```
import pandas as pd
from catboost import CatBoostClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

We perform all the same preprocessing steps as the Logistic Regression model in the last section but in this case, we do not have to impute the values since Catboost has mechanisms that already exist to deal with missing values. We also do not have to use the standard scaler method since the splits are based on the order of the values, not their absolute magnitudes. Therefore, whether a feature is scaled or not, the splits remain the same.

After repeating the steps above (dropping unnecessary columns, one-hot encoding features (embarked and sex), and setting X &y), we can finally implement the tree model. We set verbose = 0 because we don't want any outputs to the console during the model training process. We fit the training data and then create the tree which predicts the values using the X_test values.

```
tree_model = CatBoostClassifier(verbose = 0)
tree_model.fit(X_train,y_train)
y_pred_tree = tree_model.predict(X_test)
accuracy_tree = accuracy_score(y_test,y_pred_tree)
print(f"Catboost accuracy score: {accuracy_tree}")
```

```
Catboost accuracy score: 0.8100558659217877
```

Ultimately, we achieved an 81% accuracy score which is good because it could be capturing complex non-linear relationships that might be present in the factors influencing survival rates.

The difference between 81% (Tree Model) and 80.4% (Logistic Regression Model) is relatively small. This means that a linear model captures the relationships in the data. This also means that the tree model might not capture significantly more complex patterns than the linear model. Since both models are performing well, this indicates that the features we have selected are "good" features.

3. **Neural Networks**

```
import pandas as pd
import numpy as np
import seaborn as sns
from sklearn.model_selection import train_test_split
from tensorflow import keras
from sklearn.metrics import accuracy_score
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
```

Once again, we perform all the same preprocessing steps as the Logistic Regression model in the last section. One key difference from the tree model is that we impute the values since Neural Networks are designed to work with complete numerical input vectors. Missing values can disrupt the flow of information through the network, leading to errors or inaccurate predictions.

We also use the standard scaler method since Neural Networks treat all features equally. If the features have different scales, larger values can dominate the learning process, while those with smaller values might be ignored. not their absolute magnitudes. Scaling ensures that all features have a similar range, which allows the network to learn meaningful patterns from all of them.

[16]:
```
model_nn = keras.Sequential([
    keras.layers.Dense(128, activation='relu', input_shape=(X_train.shape[1],)),
    keras.layers.Dropout(0.2),
    keras.layers.Dense(64, activation='relu'),
    keras.layers.Dropout(0.2),
    keras.layers.Dense(1, activation='sigmoid')
])
model_nn.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model_nn.fit(X_train, y_train, epochs=100, batch_size=32, verbose=0)
y_pred_nn = (model_nn.predict(X_test) > 0.5).astype(int)
accuracy_nn = accuracy_score(y_test, y_pred_nn)
print(f"Neural Network Accuracy: {accuracy_nn}")
```

```
C:\Users\shaur\AppData\Local\Programs\Python\Python312\Lib\site-packages\keras\src\layers\core\dense.py:87: UserWarning: Do not pass an `input_shape`/`inpu
t_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
Neural Network Accuracy: 0.7932960893854749
```

Our Neural Network Model achieves a 79.3% accuracy score which means that our network has learned some meaningful patterns from the training data and can generalize unseen data with a good degree of success.

Niketan Baranwal

Model Comparison

Our Logistic Regression model achieved an 80.4% accuracy score
Our Tree Model achieved an 81% accuracy score
Our Neural Network Model achieved a 79.3% accuracy score

These results show us that a simple model (Logistic Regression model) performs as well as the more complex models (Tree model & Neural Network), highlighting the importance of starting with simpler models. Simpler models are preferable due to their ease of understanding and low computational cost.

The choice of the model also depends on the requirements of the specific problem.

- If interpretability is required then, in this case, the Logistic Regression model would be the best choice
- If higher accuracy is required then, in this case, the Tree Model would be the best choice
- If the dataset is much larger then, in this case, the Neural Network would outperform all the other models

All three models are effective for the Titanic dataset. The accuracies are so close to each other which indicates that the features that have been selected are very “good” features.

Ultimately, we would choose the Logistic Regression model due to its simplicity and high accuracy.