Predicting Travel Insurance Purchases Using Machine Learning

Chedi Mnif, Peter Sayegh, Karel Moryoussef

Impact and Applications of Predictive Modeling in Travel Insurance

- Targeted Advertising
  - Identify potential customers more likely to buy travel insurance.
  - Enhance marketing efficiency and conversion rates.
- Resource Allocation:
  - Efficiently allocate resources like sales teams based on conversion likelihood.
  - Optimize operational costs and improve customer service.

Our model's could for example be sold to insurance companies

Table of contents

- Data Pre-Processing
- Exploratory Data Analysis (EDA)
- KNN and Logistic Regression
- Neural Networks
- Tree Models
- Main Difficulties
- Conclusion

Data Pre-Processing

Data Acquisition and Processing

	Unnamed: 0	Age	Employment Type	GraduateOrNot	AnnualIncome	FamilyMembers	ChronicDiseases	FrequentFlyer	EverTravelledAbroad	TravelInsurance
0	0	31	Government Sector	Yes	400000	6	1	No	No	0
1	1	31	Private Sector/Self Employed	Yes	1250000	7	0	No	No	0
2	2	34	Private Sector/Self Employed	Yes	500000	4	1	No	No	1
3	3	28	Private Sector/Self Employed	Yes	700000	3	1	No	No	0
4	4	28	Private Sector/Self Employed	Yes	700000	8	1	Yes	No	0

Kaggle Dataset

<u>Dataset Specificities</u>
Total samples: 1982
Total features: 8
Target variable: Travel Insurance

No missing values (no need for imputing)

Categorical to Numerical

Features such as ‘GraduateOrNot’, ‘Employment Type’, ‘FrequentFlyer’, ‘EverTravelledAbroad’ need to be transformed into numerical values to be processed correctly.

```
#Converting categorical variables to numerical variables
data["GraduateOrNot"]= data["GraduateOrNot"].map({"No" :0, "Yes" : 1})
data["FrequentFlyer"]= data["FrequentFlyer"].map({"No" :0, "Yes" : 1})
data["EverTravelledAbroad"]= data["EverTravelledAbroad"].map({"No" :0, "Yes" : 1})
data["Employment Type"]= data["Employment Type"].map({"Government Sector" :0, "Private Sector/Self Employed" : 1})

#Unnamed column dropped from dataset
data = data.drop(columns=['Unnamed: 0'])
```

	Age	Employment Type	GraduateOrNot	AnnualIncome	FamilyMembers	ChronicDiseases	FrequentFlyer	EverTravelledAbroad	TravelInsurance
0	31	0	1	400000	6	1	0	0	0
1	31	1	1	1250000	7	0	0	0	0
2	34	1	1	500000	4	1	0	0	1
3	28	1	1	700000	3	1	0	0	0
4	28	1	1	700000	8	1	1	0	0

Data Overview: Standardization of Data

	Age	Employment Type	GraduateOrNot	AnnualIncome	FamilyMembers	ChronicDiseases	FrequentFlyer	EverTravelledAbroad	TravelInsurance
count	1987.000000	1987.000000	1987.000000	1.987000e+03	1987.000000	1987.000000	1987.000000	1987.000000	1987.000000
mean	29.650226	0.713135	0.851535	9.327630e+05	4.752894	0.277806	0.209864	0.191243	0.357323
std	2.913308	0.452412	0.355650	3.768557e+05	1.609650	0.448030	0.407314	0.393379	0.479332
min	25.000000	0.000000	0.000000	3.000000e+05	2.000000	0.000000	0.000000	0.000000	0.000000
25%	28.000000	0.000000	1.000000	6.000000e+05	4.000000	0.000000	0.000000	0.000000	0.000000
50%	29.000000	1.000000	1.000000	9.000000e+05	5.000000	0.000000	0.000000	0.000000	0.000000
75%	32.000000	1.000000	1.000000	1.250000e+06	6.000000	1.000000	0.000000	0.000000	1.000000
max	35.000000	1.000000	1.000000	1.800000e+06	9.000000	1.000000	1.000000	1.000000	1.000000

- We notice the features have different scaling which can affect the models. We want all features to contribute equally to the model.

<u>Standardization allows:</u>

Equal contribution of features

Improved model performance (especially for models such as KNN)

Standardization and Data Split

```python
# Split the training and testing data (70:30 )
training_points = data.drop(columns=['TravelInsurance'])
training_labels = data['TravelInsurance']

#Random state is to ensure the reproducibility of the results
X_train, X_test, y_train, y_test = train_test_split(
    training_points,
    training_labels,
    test_size = 0.3,
    random_state = 42)

#Normalization of Features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Data point

Mean

$$z = \frac{(x - \mu)}{\sigma}$$

Standard deviation

train_test_split: 70% training data, 30% test data

Dataset Post-Preprocessing

	Age	Employment Type	GraduateOrNot	AnnualIncome	FamilyMembers	ChronicDiseases	FrequentFlyer	EverTravelledAbroad
0	1.495933	-1.639208	0.425404	0.956849	1.373744	-0.626726	-0.523508	-0.477405
1	0.470277	0.610051	0.425404	0.823967	1.373744	-0.626726	-0.523508	-0.477405
2	-0.213494	0.610051	0.425404	0.691084	0.144581	1.595593	-0.523508	-0.477405
3	-0.555379	0.610051	0.425404	-0.903505	-1.084581	-0.626726	-0.523508	-0.477405
4	-1.239150	0.610051	0.425404	-1.169270	1.988325	-0.626726	-0.523508	-0.477405

X_train.head()

	Age	Employment Type	GraduateOrNot	AnnualIncome	FamilyMembers	ChronicDiseases	FrequentFlyer	EverTravelledAbroad
count	1.390000e+03	1.390000e+03	1.390000e+03	1.390000e+03	1.390000e+03	1.390000e+03	1.390000e+03	1.390000e+03
mean	2.300318e-16	7.412136e-17	5.750796e-17	1.303514e-16	1.277955e-16	3.322682e-17	2.555909e-17	-2.044727e-17
std	1.000360e+00	1.000360e+00	1.000360e+00	1.000360e+00	1.000360e+00	1.000360e+00	1.000360e+00	1.000360e+00
min	-1.581035e+00	-1.639208e+00	-2.350707e+00	-1.700800e+00	-1.699162e+00	-6.267261e-01	-5.235079e-01	-4.774046e-01
25%	-5.553793e-01	-1.639208e+00	4.254040e-01	-9.035050e-01	-4.699998e-01	-6.267261e-01	-5.235079e-01	-4.774046e-01
50%	-2.134939e-01	6.100507e-01	4.254040e-01	-1.062104e-01	1.445813e-01	-6.267261e-01	-5.235079e-01	-4.774046e-01
75%	8.121622e-01	6.100507e-01	4.254040e-01	8.239668e-01	7.591624e-01	1.595593e+00	-5.235079e-01	-4.774046e-01
max	1.837818e+00	6.100507e-01	4.254040e-01	2.285674e+00	2.602906e+00	1.595593e+00	1.910191e+00	2.094659e+00

X_train.describe()

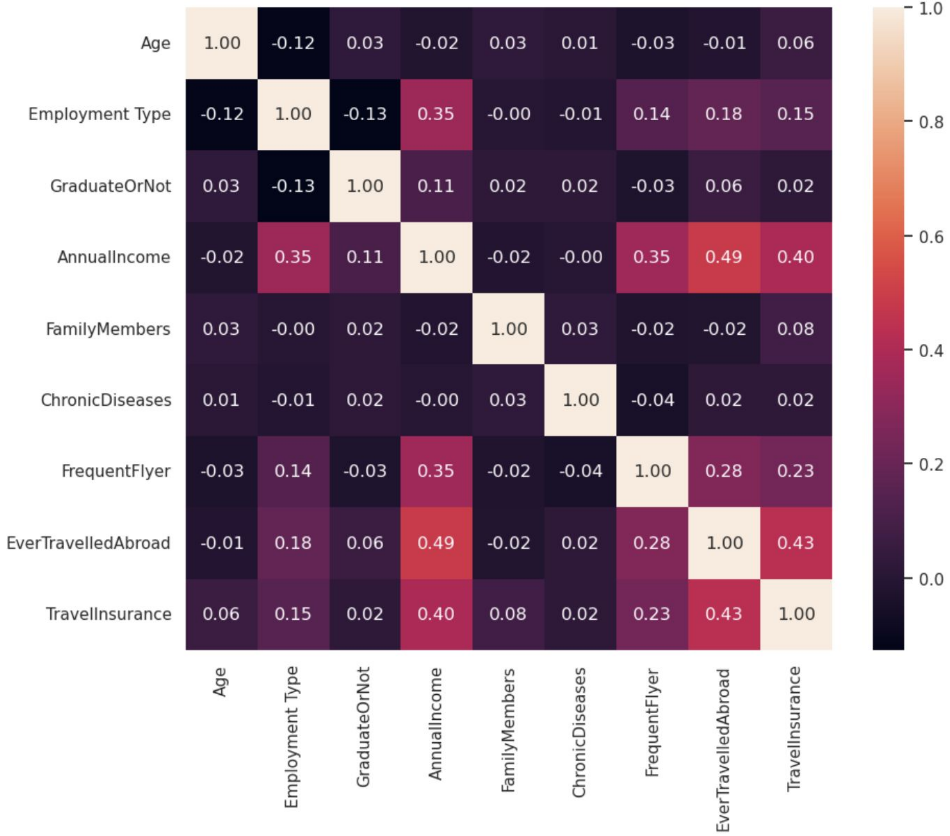
Exploratory Data Analysis

Imbalanced Dataset

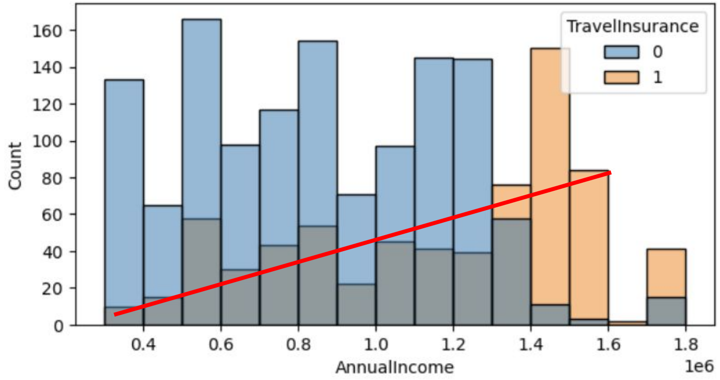
- Imbalance between Yes and No Instances
- Tried resampling techniques (SMOTE,manual, undersampling), however didn’t lead to good results
- Consequently, it is natural that the models used will ‘learn’ the ‘No instances’ better.

Correlation Matrix

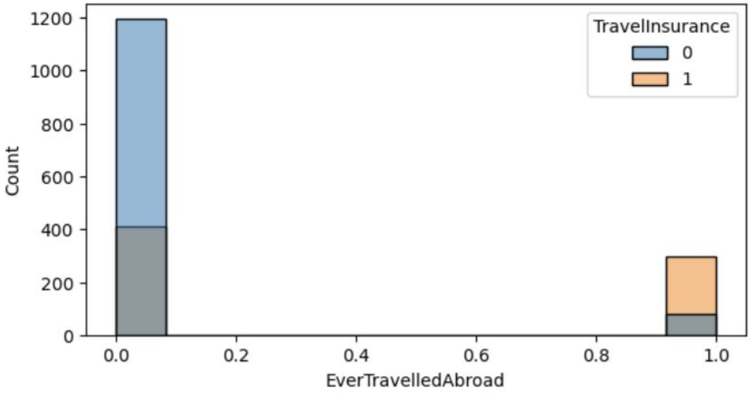
- Features with highest correlation to target variable
  - Annual Income (0.49)
  - Ever Travelled Abroad (0.43)
  - Frequent Flyer (0.23)
- The rest of the variables are not highly correlated to the target variable



Best ‘Predictors’ for Travel Insurance



- Notice that most people with high annual income have travel insurance which highlights the high correlation



- Most people who have already travelled abroad have travel insurance

Basic Models: Logistic Regression and KNN

Logistic Regression

```
#We perform simple Logistic Regression
classifier = LogisticRegression(random_state=4)
classifier.fit(X_train, y_train)
predictions = classifier.predict(X_test)
accuracy = accuracy_score(y_test, predictions)
print(accuracy)
```

```
0.7487437185929648
```

	precision	recall	f1-score	support
0	0.78	0.91	0.84	375
1	0.79	0.57	0.66	222
accuracy			0.78	597
macro avg	0.78	0.74	0.75	597
weighted avg	0.78	0.78	0.77	597

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

K-Nearest Neighborhoods (KNN) w/different metrics

- Experimented with the different distance metrics - Euclidean, Manhattan, Chebyshev

```
Accuracy results for different distance metrics:
euclidean: 0.7805695142378559
manhattan: 0.7705192629815746
chebyshev: 0.7470686767169179

Distance Metric: euclidean
Accuracy: 0.7805695142378559
              precision    recall  f1-score   support

           0       0.77      0.93      0.84       375
           1       0.82      0.52      0.64       222

    accuracy                           0.78       597
   macro avg       0.80      0.73      0.74       597
weighted avg       0.79      0.78      0.77       597
```

1. Euclidean Distance

Formula:

$$d(p, q) = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2}$$

2. Manhattan Distance (L1 norm)

Formula:

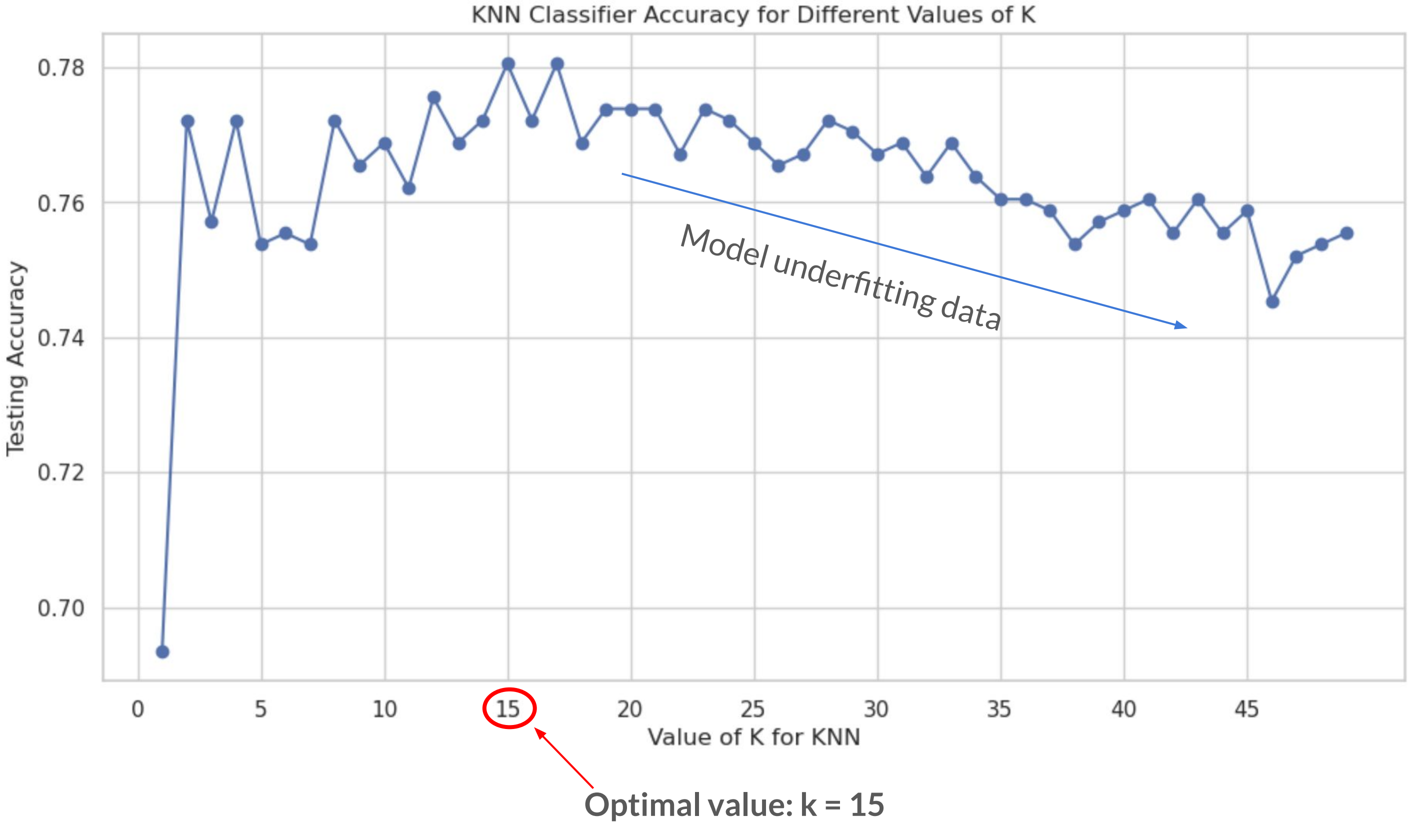
$$d(p, q) = \sum_{i=1}^{n} |p_i - q_i|$$

3. Chebyshev Distance (L∞ norm)

Formula:

$$d(p, q) = \max_i(|p_i - q_i|)$$

Choosing k - Number of Neighbors



Deep Neural Networks

Neural Network Architecture

```
# Build the Neural Network
model = Sequential()
model.add(Dense(32, input_dim=X_train.shape[1], activation='relu'))
model.add(Dense(16, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

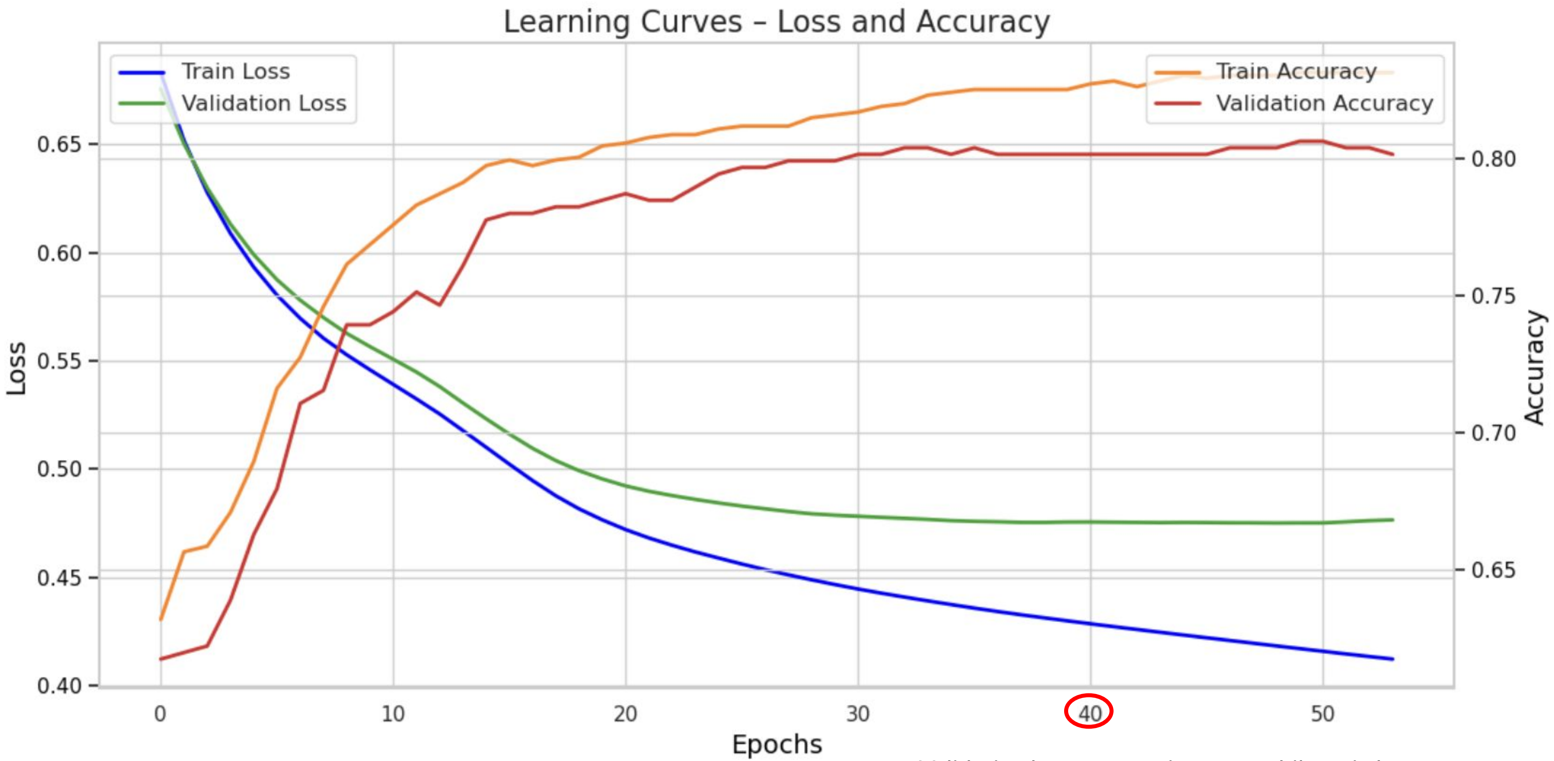
Input layer: Standardized features as input
1st hidden layer: 32 neurons (relu activation)
2nd hidden layer: 16 neurons (relu activation)
3rd hidden layer: 8 neurons (relu activation)
Output layer: 1 neuron (sigmoid activation) - probability that input will purchase travel insurance
Loss: binary-cross entropy - Optimizer: adam

Optimization of parameters: Early Stopping

- Early stopping: stops the training when the validation loss starts to reverse back up (patience = 5)
- Used to prevent overfitting of the data
- Validation split:
  - at each epoch, model is trained on 70% of the training data
  - model is tested on 30% of the training data
- restore_best_weights = True - used to restore the weights that minimize the validation loss

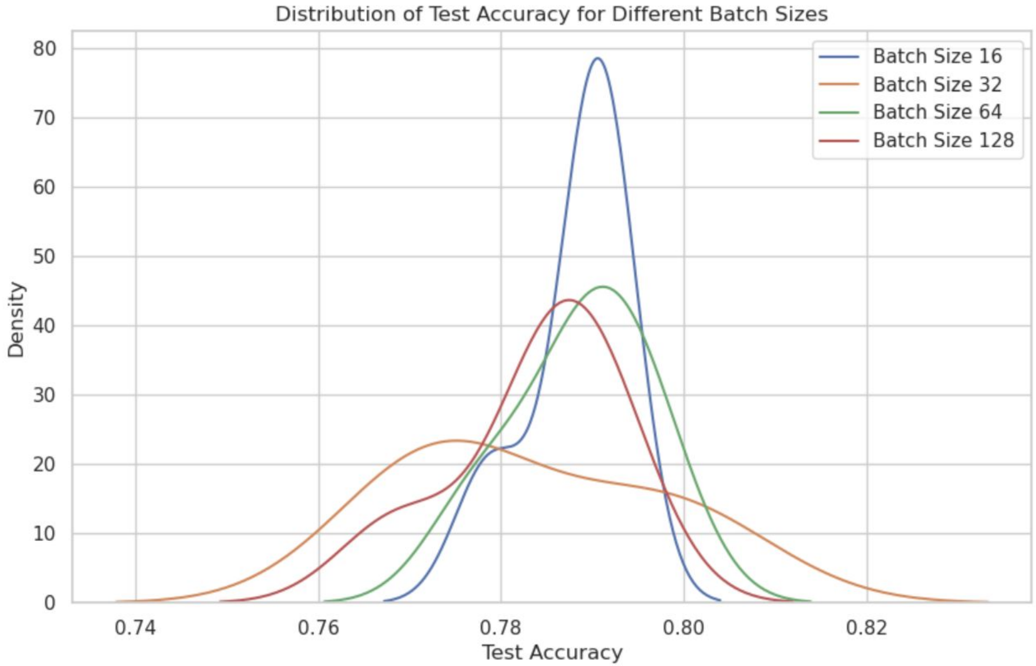
```
# Early stopping callback
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

# Train the model with early stopping
history = model.fit(X_train, y_train, epochs=100, batch_size=16, validation_split=0.3, verbose=1, callbacks=[early_stopping])
```

Learning Curves Loss/Accuracy



Validation loss starts to increase while train loss continues to decrease = overfitting

Optimization of parameters: Batch size



- We can also optimize the batch size - ran the model 10 times (took a bit of time due to computational power)

- Test each time for different batch sizes [16,32,64,128]

Hence, we chose an optimal batch size of 16 (stochastic gradient descent) also making it more computationally efficient.

Conclusion Neural Networks

- In the end, we obtained the following results with optimized parameters

```
Test Loss (Binary Cross-Entropy): 0.4979952275753021
Test Accuracy: 0.7922948002815247
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
Classification Report:
              precision    recall  f1-score   support

           0       0.79      0.91      0.85       375
           1       0.79      0.60      0.68       222

    accuracy                           0.79       597
   macro avg       0.79      0.75      0.76       597
weighted avg       0.79      0.79      0.78       597

Accuracy: 0.7922948073701842
```

Improved recall (compared to KNN and LR)

Decision Trees

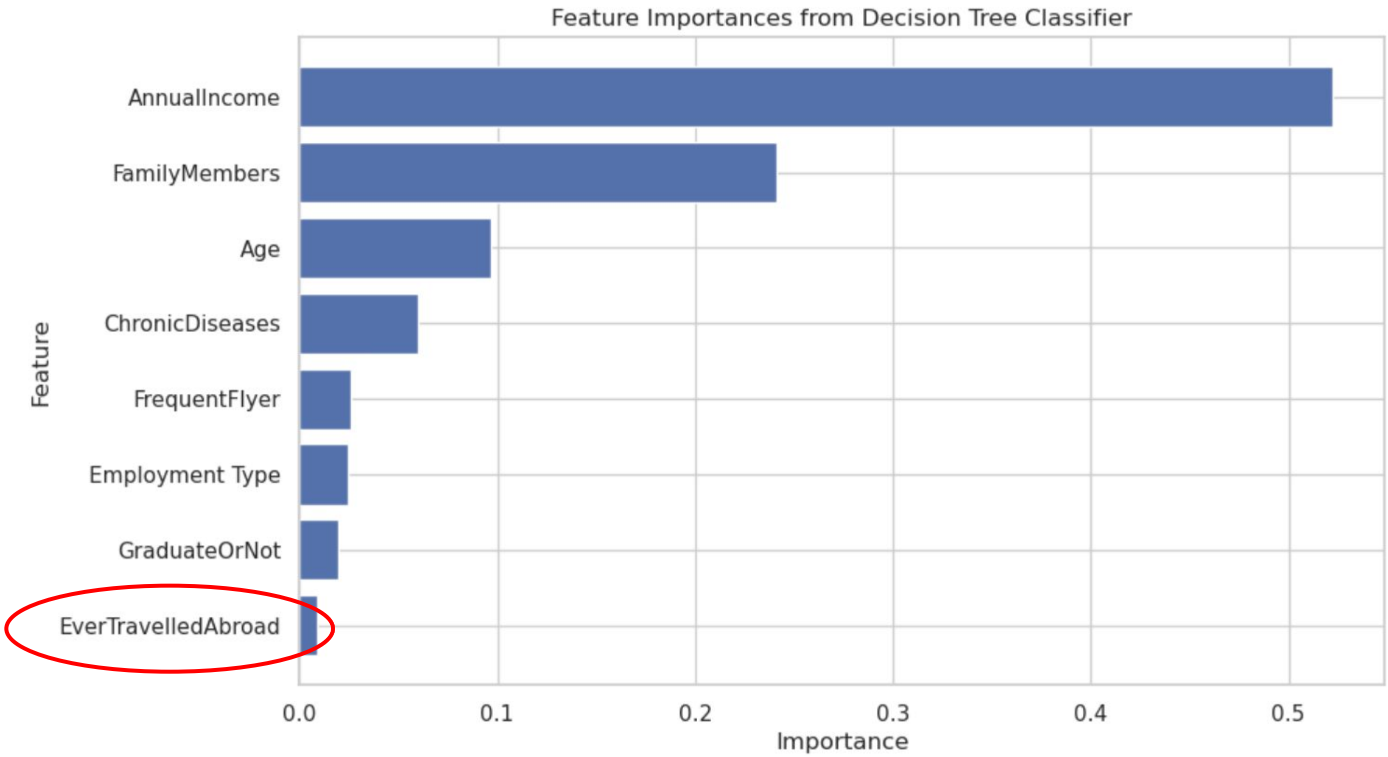
Basic Decision Tree Classifier

- Begin by making a simple decision tree without tuning
    - Disadvantages: overfitting - untuned decision trees expand until the final leaves are pure (entropy = 0 or gini-entropy = 0)
        - Captures noise in the data → overfitting the data
- Not an ideal model (can have a lot of variance if tested on new data)
- Testing on both gini entropy and classic entropy

```
# Train a Decision Tree Classifier
clf = DecisionTreeClassifier(random_state=42,criterion='entropy')
clf.fit(X_train, y_train)
```

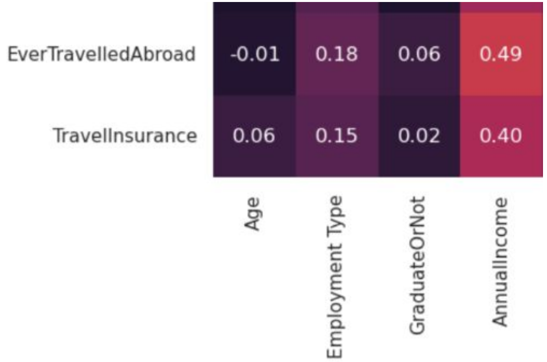
```
Accuracy: 0.7721943048576214
Classification Report:
              precision    recall  f1-score   support

           0       0.79      0.87      0.83       375
           1       0.74      0.60      0.66       222

    accuracy                           0.77       597
   macro avg       0.76      0.74      0.75       597
weighted avg       0.77      0.77      0.77       597
```

Feature Importance (decision tree classifier)



‘EverTravelledAbroad’ Least Important Feature

- Intuitively, we might expect "EverTravelledAbroad" to be a significant predictor of whether a person buys travel insurance, as past travel experience could increase the likelihood of purchasing insurance for future trips.
- Low information gain when splitting with annual income and then with ever travelled since correlation is high (brings similar information)
- "AnnualIncome" is highly correlated with "EverTravelledAbroad" and provides strong splits, the model uses "AnnualIncome" more often, reducing the apparent importance of "EverTravelledAbroad".



HyperParameter Tuned Decision Tree

```python
# Define the parameter grid
param_grid = {
    'max_depth': [3, 5, 7, 10, None],
    'min_samples_split': [2, 5, 10, 20],
    'min_samples_leaf': [1, 2, 5, 10],
    'max_features': [None, 'sqrt', 'log2'],
    'criterion': ['gini', 'entropy']
}

# Initialize the GridSearchCV
grid_search = GridSearchCV(estimator=DecisionTreeClassifier(random_state=42),
                           param_grid=param_grid,
                           scoring='accuracy',
                           cv=5,
                           n_jobs=-1,
                           verbose=1)
```

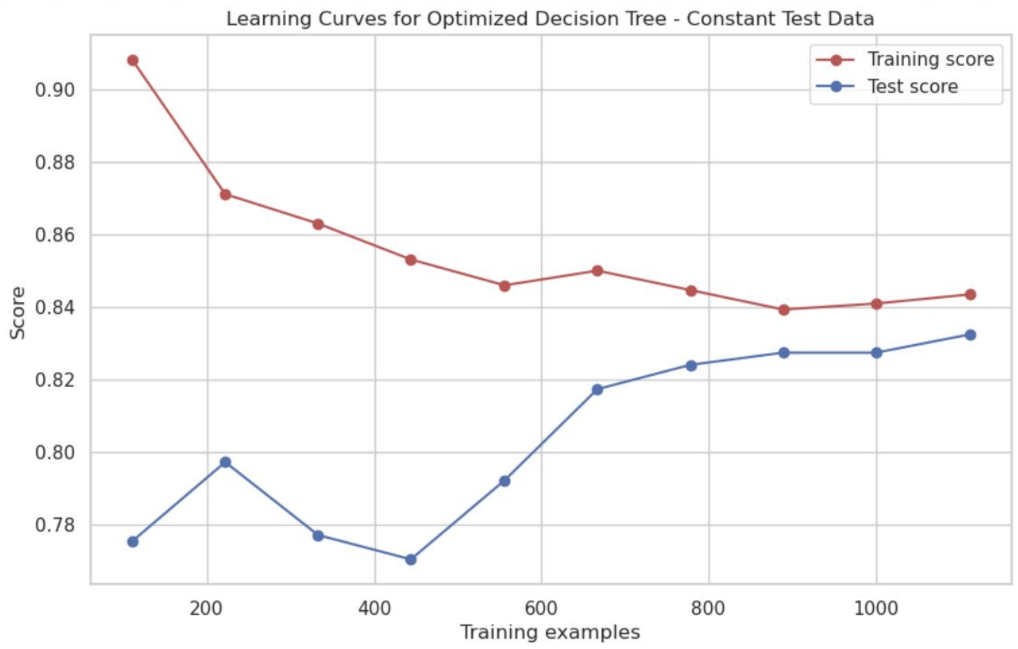
```
Test Accuracy (Best Model): 0.830820770519263
Classification Report (Test Data, Best Model):
              precision    recall  f1-score   support

           0       0.80      0.98      0.88       375
           1       0.94      0.59      0.72       222

    accuracy                           0.83       597
   macro avg       0.87      0.78      0.80       597
weighted avg       0.85      0.83      0.82       597
```

- Gridsearch to test for different parameters

{'criterion': entropy, 'max_depth': 5, 'max_features': None, 'min_samples_leaf': 1, 'min_samples_split': 2}

<u>We also notice that the metrics are balanced, good f1 scores for both 0 and 1</u>

Learning Curves for Parametrized Decision Tree



- Less training examples → overfitting and worst accuracy on test data
- As we increase the number of training examples, variance of result decreases

We notice a convergence of accuracy towards 84% for both training test and test data set

Ensemble Methods: Bagging Classifier

Method that consists in splitting the training set into subsets and training a classifier on each subset (then taking the majority vote for the output)

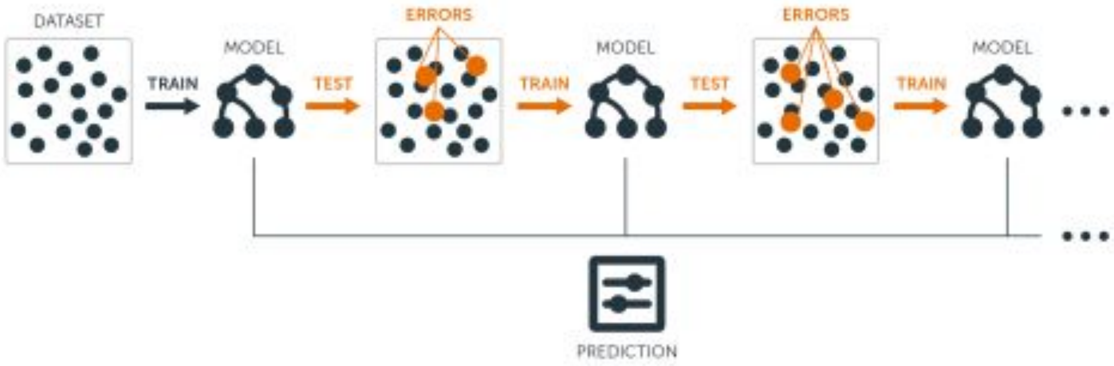
```
Test Accuracy (Best Bagging Model): 0.8174204355108877
Classification Report (Test Data, Best Bagging Model):
              precision    recall  f1-score   support

           0       0.79      0.98      0.87       375
           1       0.93      0.55      0.69       222

    accuracy                           0.82       597
   macro avg       0.86      0.76      0.78       597
weighted avg       0.84      0.82      0.80       597
```

Hyper-parameter tuned bagging classifier results

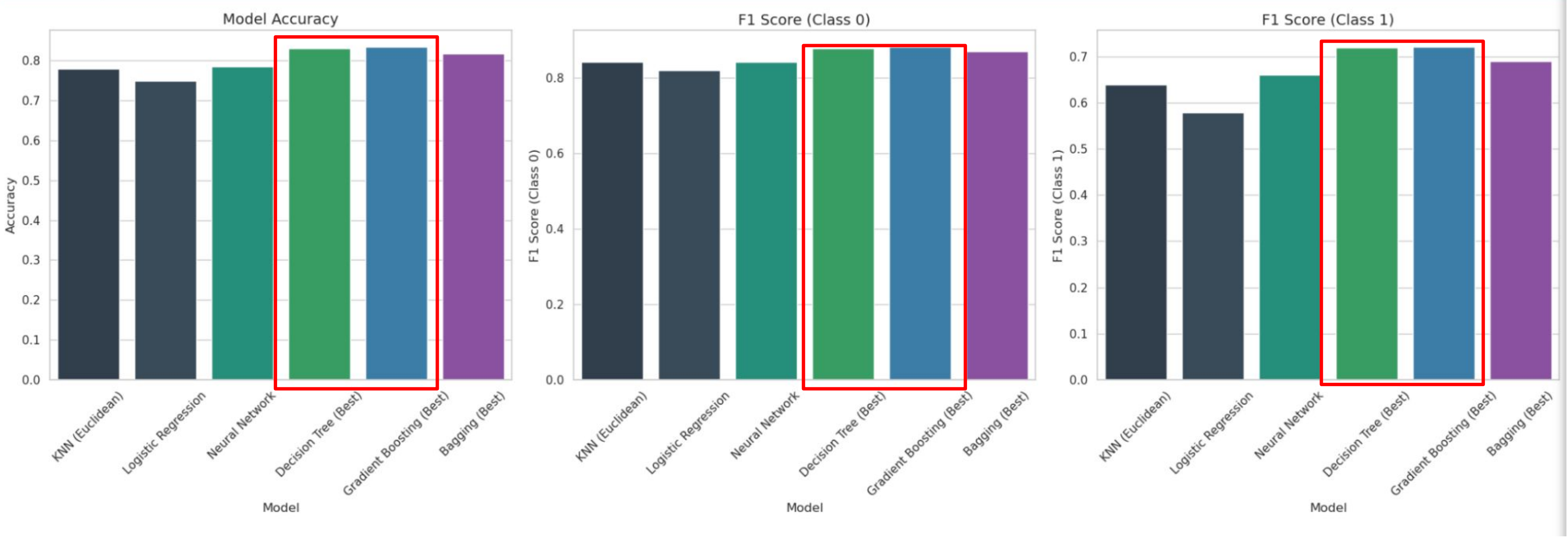
Ensemble Methods: Gradient Booster

```
Test Accuracy (Best Gradient Boosting Model): 0.8341708542713567
Classification Report (Test Data, Best Gradient Boosting Model):
              precision    recall  f1-score   support

           0       0.80      0.99      0.88       375
           1       0.96      0.58      0.72       222

    accuracy                           0.83       597
   macro avg       0.88      0.78      0.80       597
weighted avg       0.86      0.83      0.82       597
```

- Balanced metrics and good f1 score for both 0 and 1 (similar results to decision tree)
- More computationally intensive



Final Results



Decision Tree and Gradient Boosting provide the best accuracies and most balanced metrics (however, gradient boosting is more computationally inefficient)

Main Challenges and Overview

- Difficulty with data set, was imbalanced which explains the low recall on 1s
  - Adding more data-points for ‘travel-insurance’ minority class would most likely increase accuracy considerably.
- Didn’t use unsupervised learning techniques such as PCA or Autoencoders, since feature-space was not that large
  - Not much interest in reducing feature space since pretty low already
- Our best performing model was our gradient boosting model and hyper-parameter tuned decision tree which provided very same results
  - 83.4% accuracy and balanced metrics
    - Travel Insurance Class: F1 Score: 72%
    - No Travel Insurance Class: F1 Score: 88%