1 Assignment 1: Predictive Modeling on Tabular Data

In this assignment, we construct a predictive model to predict churn of Telco customers based on a variety of collected metrics and evaluted on profit @ top-20, performance score, and AUC. The end goal is to predict which customers will churn and which will stay. Predicting this can be a challenge if the number of churners significantly differs from those staying. For example, say only 5% of customers churn. Then, the easiest (and laziest) way to predict what a customer will do is to guess that he'll stay, as you will be correct 95% of the time. So, one wishes to accurately identify the small number of churning customers while accurately maintaining who is staying.

1.1 Data Preprocessing and Exploration

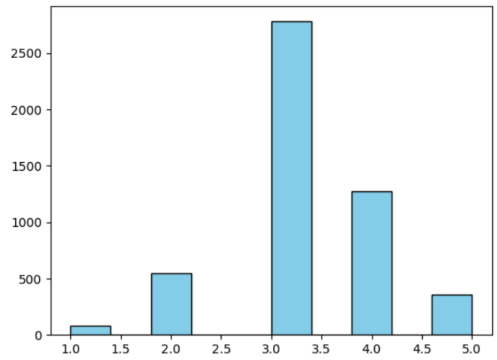
The data consists of recorded information of Telco customers. 40 features are measured: 11 categorical and 29 continuous. Basic features include gender, age, type of handset, peak calling minutes, national call minutes, and cost. One feature - "connect date" was transformed from its connect date to another feature "use year" to quantify how long the customer has been with the company. "Gender","High Dropped calls" and "No Usage" are encoded as binary feature, the features "tariff" and "handset" were one-hot encoded for analyzing the categorical data. The "Usage band" shows an ordinal tendency, but after we explored the distribution of different levels, we decided to apply the percentile ranking to adapt the distribution.

```
# encoding the ordinal features-first integer
Mapping_ub={'Med': 3, 'MedLow': 2, 'MedHigh': 4, 'High': 5,'Low':1}
X['Usage_Band']=X['Usage_Band'].replace(Mapping_ub)
```

```
/var/folders/d4/cp4g0mns44s67403lg_133gw0000gn/T/ipykernel_55513/735085468.py:3: FutureWarning: Downcasting behavi
  X['Usage_Band']=X['Usage_Band'].replace(Mapping_ub)
```

```
plt.hist(X['Usage_Band'], bins=10, color='skyblue', edgecolor='black')
```

```
(array([  84.,    0.,  547.,    0.,    0., 2779.,    0., 1270.,    0.,
         360.]),
 array([1. , 1.4, 1.8, 2.2, 2.6, 3. , 3.4, 3.8, 4.2, 4.6, 5. ]),
 <BarContainer object of 10 artists>)
```




```
# encode it as the percentile ranking
ub_ran=X['Usage_Band'].value_counts(normalize=True)*100
X['Usage_Band_perank']=X['Usage_Band'].map(ub_ran)
```

Figure 1: Percentile Ranking

We choose feature in a cautious way, to make sure the distribution and outliers are not so exaggerated. Also we check if there were any feature strongly correlated to each other.

```
# for the numerical features
len(nume_features)
nume_features.remove('target')
print(nume_features)
```

```
['Age', 'L_O_S', 'Dropped_Calls', 'Peak_calls_Sum', 'Peak_mins_Sum', 'OffPeak_calls_Sum', 'OffP
```

```
#check the outliers
selected_columns = X_train.iloc[:, list(range(3, 27)) + list(range(28, 33))]
# Visualize outliers using box plots
for col in selected_columns.columns:
    plt.figure(figsize=(8, 6))
    sns.boxplot(x=selected_columns[col])
    plt.title(f'Box plot of {col}')
    plt.xlabel(col)
    plt.show()
```

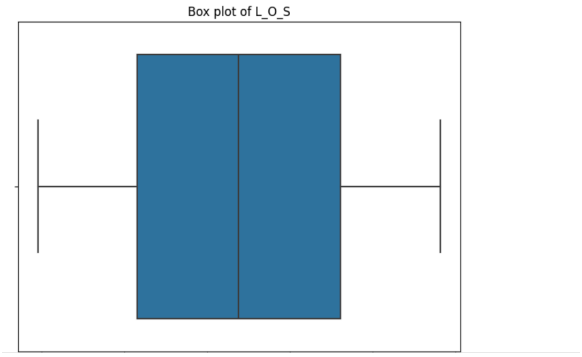


Figure 2: Distribution and Outliers

Finally we choose 22 features to fit the model.

```
features_corr=X_train[X_features].corr()

# Plot heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(features_corr, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Heatmap')
plt.show()
```

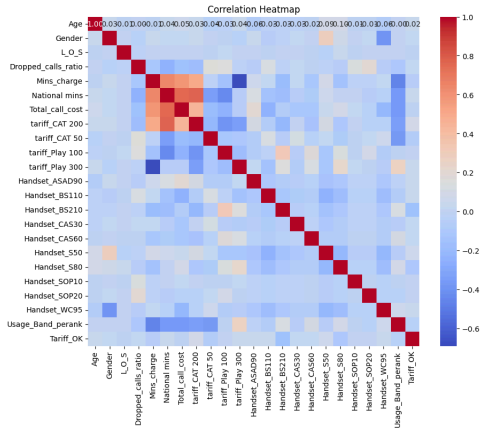


Figure 3: Correlation Heatmap

After confirming the features, the data was split into a simple random training/test set on a 80/20 split. Rows with missing values were dropped.

1.2 Model Fitting

The top-20 and AUC are mainly used to evaluate the model, therefore, we define a function to create the top-20 metric with the probability where the true positive has the highest 20th average cost min.

Four main models are shown in this part.

```
# top 20 metric
customer_costs = X_test['average cost min']
def profit_top_20(y_probs):
    # Sort the predicted probabilities and get indices of top-k predicted churners/1
    true_positives = np.where(y_test == 1)[0]
    sorted_probs = y_probs[true_positives]
    top_20 = true_positives[(-sorted_probs).argsort()[:20]]
    profit_top_20 = customer_costs.iloc[top_20].sum()
    return profit_top_20
```

Figure 4: Top-20 Metric

1.2.1 Random Forest

The first model for consideration is a simple random forest. The random forest generates a number of trees, each trees takes a set of observations, then each tree chooses a random subset of the features at each split, chooses the best feature for that split, and then averages the results to predict the categorization.

The initial random forest was trained with 400 trees and all the features. Then the forest was trained, evaluated, and the feauture importance explored. The initial forest had an AUC for a ROC curve of 0.69. The most important features were ”dropped calls ratio”, ”usage band”, and ”age”. A second forest was made using the 5 most important features in an attempt to obtain a more simple model. However, the performance decreased to AUC of 0.67. The overall performance was low. After that, we try all the final features chosen, which shows a stable and better results of the top-20 and AUC.

1.2.2 Extremely Random Forest

Next, an extremely random forest was trained for the data. The main difference between the random forest and the extremely random forest is the extremely random forest chooses a random value to split on the node (for a set of features) instead of choosing the ”best” value to split on. The main advantage of the Extremely Random Forest is a reduction in variance. This is

rather important for this task as we want to be as specific as possible in the predictions. Further, the data is not bootstrapped by default.

For this model, we use a smaller set of features than for the random forest, as some features were shown to be less than important. The forest was initialized with 1000 trees. Performance was evaluated by 10-fold cross-validation. The average CV performance on the training set was 0.919 ($sd = 0.014$), and for the test set 0.908 ($sd = 0.027$). The AUC for the model was 0.82 on the training set and 0.90 on the test set. These results fall in line with what we would expect for testing/training performance. The results are not so high as to suggest overfitting but not so low that they are useless. Figure 5 shows the feature importances from the resulting tree.

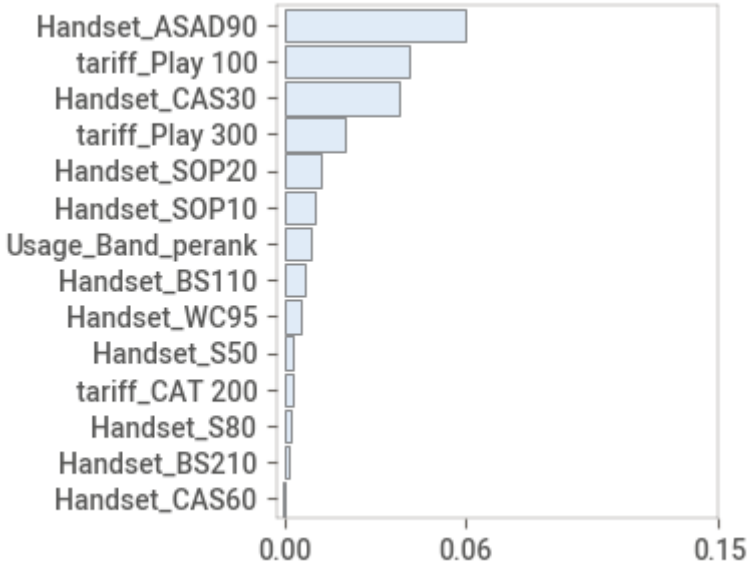


Figure 5: Feature Importances from Extremely Random Trees

Taking the top 3 features, another extremely random forest model is constructed. This time, the performance decreases slightly with an accuracy score of 0.89 and a drop of AUC to 0.71. While it may be useful to drop certain features, having a high AUC of ROC is important to this task, so it is suggested to keep the features even if they appear unimportant.

1.2.3 XGBoost

XGBoost was used next to train another model. XGBoost creates trees iteratively, correcting for errors along the way and using L1 and L2 regularization for feature selection. Performance was measured on the same metrics. Cross-validation performance was 0.91. Prediction scores (accuracy) resulted in 0.89. AUC was 0.92.

1.2.4 KNN

KNN was used for prediction. The initial model is compared to the $k = 5$ nearest neighbours. The AUC of KNN was 0.84 indicating a fairly decent performance. Raising $k = 10$ increased AUC to 0.87. Raising $k = 25$ further increased AUC to 0.89. While it may be tempting to continually increase k, increasing it may lead to too low variance and higher bias.

```
neighbors = np.arange(1, 21)
train_accuracies = {}
test_accuracies = {}

for neighbor in neighbors:
  knn = KNeighborsClassifier(n_neighbors=neighbor)
  knn.fit(X_train[X_features], y_train)
  train_accuracies[neighbor] = knn.score(X_train[X_features], y_train)
  test_accuracies[neighbor] = knn.score(X_test[X_features], y_test)
```

```
plt.plot(neighbors,train_accuracies.values(), label="train_accuracies")
plt.plot(neighbors, test_accuracies.values(), label="test_accuracies")
plt.legend()
plt.xlabel("Number of Neighbors")
plt.ylabel("Accuracy")
plt.show()
```

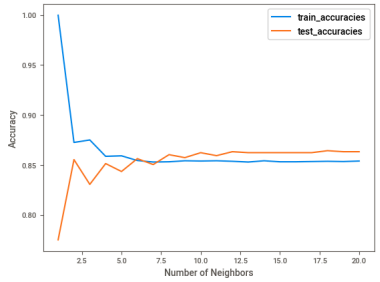


Figure 6: KNN

1.3 Model Comparison

Five model with default parameters are compared to get the pure evaluation.

Models	Top-20	AUC
Random Forest	3.60	0.93
XG Boost	3.73	0.92
KNN	4.77	0.61
Ada boost	3.83	0.92
Logistic Regression	3.83	0.92

Table 1: Model Comparison with Default Parameters

Combine the table 1 and the figure 7, the random forest seems more robust also with the higher AUC value.

```python
## training model
models={ "KNN": KNeighborsClassifier(),"Random Forest": RandomForestClassifier(),
        "XGBoost": XGBClassifier(),"ADA Boost": AdaBoostClassifier() }
results= []
for model in models.values():
      kf=KFold(n_splits=6, random_state=42, shuffle=True)
      cv_results=cross_val_score(model, X_train[X_features], y_train, cv=kf)
      results.append(cv_results)
plt.boxplot(results, labels=models.keys())
plt.show()
```

```
/Users/hy/StaDS/23-24 spring/Advanced analytics in a big data world/advanced1/lib/python3.9/site-packages/s
  warnings.warn(
/Users/hy/StaDS/23-24 spring/Advanced analytics in a big data world/advanced1/lib/python3.9/site-packages/s
  warnings.warn(
/Users/hy/StaDS/23-24 spring/Advanced analytics in a big data world/advanced1/lib/python3.9/site-packages/s
  warnings.warn(
/Users/hy/StaDS/23-24 spring/Advanced analytics in a big data world/advanced1/lib/python3.9/site-packages/s
  warnings.warn(
/Users/hy/StaDS/23-24 spring/Advanced analytics in a big data world/advanced1/lib/python3.9/site-packages/s
  warnings.warn(
/Users/hy/StaDS/23-24 spring/Advanced analytics in a big data world/advanced1/lib/python3.9/site-packages/s
  warnings.warn(
```

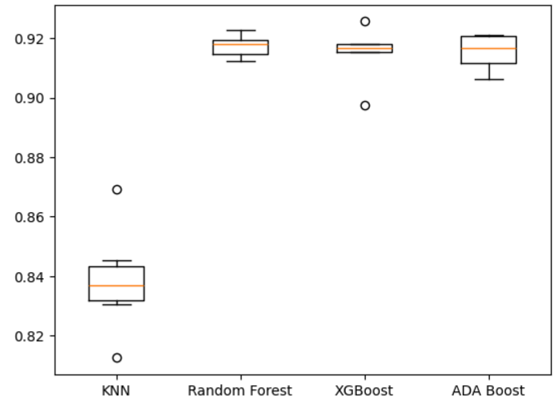


Figure 7: Model Comparison

1.4 Reflection about Submitted Model

The random forest after tuning the hyperparameters is chosen to submit on the leaderboard with the help of the random research.

```
seed=100
rf=RandomForestClassifier(n_estimators=476,
                          max_features = "log2",
                          bootstrap = True,
                          class_weight ='balanced',
                          random_state = seed,
                          max_depth=14,
                          min_samples_split=14,
                          min_samples_leaf=6)
rf.fit(X_train[X_features],y_train)
```

Figure 8: Submitted Model

We check the confusion matrix and calculate the precision and recalls, the precision is approximately 0.94 and the recall is 0.78. It is good at predicting the outcomes as the positive but with the recalls has a little lower value , incorrectly detecting the false one.

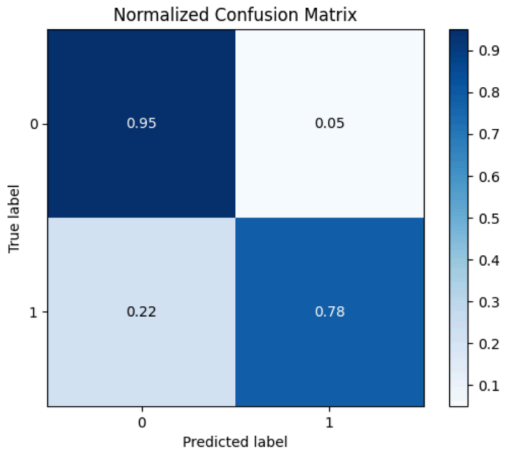


Figure 9: Normalized Confusion Matrix

We submit the model considering two metrics of top-20 and the AUC. The top-20 is at prior because of the profitability. Targeted to the customer, we are thinking how to make the cost tightly connected with the probability. So after we get the probability, we use a small trick to weight the probability with the ”average cost min”, which in our opinion can offset the misclassification in some degree.

```
# Calculate proportional prediction
test['pred'] = probs_final*test['average cost min']

print(test['average cost min'])
```

```
0       0.116924
1       0.120457
2       0.149435
3       0.137372
4       0.180704
          ...
1677    0.122342
1678    0.163502
1679    0.127485
1680    0.205830
1681    0.110151
Name: average cost min, Length: 1682, dtype: float64
```

Figure 10: Probs with Average Cost min

After releasing the result of the leaderboard, we confirm that the cost is an important index to evaluate the customer churn. It can be very valuable when making decisions. On the public leaderboard the result shows less fit than the hidden one, which makes us to think more about the generalization of the model. Compared between the public and hidden test results, it indicates the balance of generalization and targeted-aim. On one hand, the model should satisfy the result of the universal metrics and the specific metric. On the other hand, the predictions also need to be case-specific to different data distributions

Final Leaderboard

This leaderboard is calculated using the hidden half of the test data. Congrats to all participants!

Compare your results with the original public leaderboard standings and reflect accordingly in your report.

	Group	Score	Public Score	Secondary Score	Secondary Public Score	Entries
1 ▲	34	5.38789	6.954977	0.952703	0.937045	14
2 ▲	7	5.166857	6.177652	0.959268	0.94188	55
3 ▲	3	5.164836	6.686506	0.939424	0.916758	57
4 ▲	32	5.089212	5.243843	0.935015	0.891194	2
5 ▲	31	5.004224	5.892537	0.950685	0.918902	22
6 ▼	99	4.999621	7.045099	0.898127	0.834821	4

Figure 11: Leaderboard Rankings (Team 32)

1.5 Looking Back to Look Ahead

Predictive models are used everyday by everyone. Whether its deciding the best way to commute to work, what you should text your significant other so they aren't so upset with you, deciding churn, or predicting what colour a pixel should be, predicting and deciding is an unavoidable task in life, one now automated to consider thousands of variables, millions of branches, and countless paths. The paths through these forests can be long, winding, and even random. We can follow the breadcrumbs back up to the root, but growing means to follow the branches

down to the end and ”branch out” from there. When we look back at the path we took we might be able to guess where the path is heading next, but the only way to know for sure it to look ahead.

Look to new a new branch, a new tree, a new path, a new future.