Team Members:

1. IMT2021007 -Sunny Kaushik
2. IMT2021087 –Shrey Salaria
3. IMT2021530 –Akash Perla

Dataset Description and Analysis:

In this project, we are provided with a dataset related to a public bus transport management system. The dataset encompasses crucial information pertaining to bus operations, specifically focusing on predicting whether a bus will arrive on time. The dataset includes essential columns such as Index, Bus ID, Departure Time, Journey Time, Bus Operator, Departure Bus Stop, Arrival Bus Stop, and the day of travel. The objective of this project is to develop a predictive model capable of determining whether a bus is likely to be on time or not based on the provided features. The target variable for this prediction task is binary, where a value of 1 denotes that the bus will arrive on time, and a value of 0 indicates otherwise. To facilitate this predictive modeling task, we have been given two key datasets: the training dataset to train our models and the test dataset to evaluate their predictive performance.

Encoding Methods:

1. Now, since some of the columns had values in non-numeric format therefore, we need to use encoding for the non-numeric data. We first went with the label encoding as there were categorical variables with ordinal relationships as Journey_Time, Day.
2. Now, since the variation of the Journey_time was quite large, and ordinal relationships were already existing therefore applied Target encoding.

Data Augmentation:

In classification problems with imbalanced classes, where some classes have significantly fewer samples than others, data augmentation can help balance the class distribution. Data augmentation can act as a form of regularization. It helps prevent overfitting by exposing the model to a wider range of variations and noise in the data. This can lead to a more generalizable model. Overfitting case was observed as the testing dataset showed an increased accuracy, but this was not the case with testing dataset (Kaggle). Therefore, we augmented the training dataset with 5000 samples of augmented rows adding noise to

Journey_time specifically. Since we have used 99% of the training dataset hence there were chances of overfitting.

Section-1 : SVM

Pre-processing:

- Removed rows with null values in the train data.
- Replaced rows with mean values in the test data.
- Applied feature encoding on Bus_ID, Bus_Operator, Departure_Bus_Stop, Arrival_Bus_Stop, Day
- Dropped non numeric columns Index, Target, DepartureTime from training data.
- Dropped non numeric columns Index, DepartureTime from test data.
- Applied standard scaling to the training and test data.

Conclusion:

SVM is computationally expensive (took 2-3 hours to run) since the size of the dataset is quite large. Also the accuracy is quite less as compared to neural networks accuracy since neural networks have high parameter capacity, allowing them to capture complex relationships in the data.

The RBF kernel can handle cases where the decision boundary is highly irregular i.e. where the classes are not easily separable making it more adaptable to a wide range of data distributions.

This kernel implicitly maps the input data to a higher-dimensional space without explicitly computing the transformed features making it computationally efficient. Polynomial/Linear kernels, on the other hand, explicitly compute the feature mapping, and the computational cost can increase with higher degrees.

In general, the RBF kernel performs the best among the 3.

Section-2 : Non-neural ensembles

Method-1 : Bagging and Boosting

In the boosting method we have used the Gradient Boosting and Xtreme Gradient Boosting (known as XGBoost classifier).

Bagging and boosting are both ensemble techniques used in machine learning to improve the performance of predictive models by combining multiple individual models or weak learners. These methods aim to reduce variance, enhance model robustness, and improve predictive accuracy.

Bagging (Bootstrap Aggregating):

Bagging involves training multiple instances of the same learning algorithm on different subsets of the training data. The subsets are randomly sampled with replacement (bootstrap sampling). Each model is trained independently, and their predictions are aggregated (e.g., averaged in regression or majority vote in classification) to make the final prediction. Bagging helps reduce overfitting by providing more diversity among the models and improving stability and generalization.

Boosting:

Boosting is an iterative ensemble method that builds multiple weak learners sequentially, where each subsequent learner focuses on the mistakes made by the previous ones. Boosting algorithms assign higher weights to misclassified instances, allowing the subsequent models to pay more attention to these instances. Models are combined in a way that emphasizes the strengths of individual models, gradually improving predictive performance. Gradient Boosting, AdaBoost, and XGBoost are popular boosting algorithms used for various machine learning tasks.

1. In our project with the above described dataset, I have applied an XGBoost classifier (tree) is used as the base estimator for a BaggingClassifier. The BaggingClassifier is configured to create an ensemble of 20 XGBoost models (due to n_estimators=20). The code fits this ensemble on the training data (X_train and Y_train) and generates predictions (y_pred) for the test data (X_test). This ensemble applies the bagging technique by training multiple instances of the XGBoost model on different subsets of the data and combining their predictions.
2. The number of estimators has been best predicted using the grid search CV method to attain the maximum accuracy.
3. The base classifier was varied with different classifier techniques including decision trees, random forest, but the best result was obtained while using XGB classifier.

In essence, applying Bagging with XGBoost as the base estimator can lead to improved performance and robustness, particularly in reducing overfitting and enhancing stability, although it might come at the cost of increased computational complexity.

```
from sklearn.ensemble import BaggingClassifier
tree=XGBClassifier(
   n_estimators=280,
   max_depth=9,
   learning_rate=0.1,
   min_child_weight=1,
   subsample=0.8,
   colsample_bytree=0.8,
   gamma=0,
   reg_alpha=0.01,
   reg_lambda=1,
   objective='binary:logistic'
)
models_ensemble['Bagging']=BaggingClassifier(base_estimator=tree,
n_estimators=20,random_state=23)
models_ensemble['Bagging'].fit(X_train,Y_train)
y_pred = models_ensemble['Bagging'].predict(X_test)
```

Here, while doing hyperparameter tuning we have used the parameters finely tuned in the First part of the project. Hence the XGB Classifier is a modified one with customised parameter tuning.

Method-2 : Voting Classifier

The Voting Classifier in scikit-learn is an ensemble method that combines multiple individual machine learning models to make predictions. It aggregates the predictions from different models and outputs the class with the most votes (for classification) or the average prediction (for regression), effectively acting as a meta-estimator.

There are two types of voting strategies in a Voting Classifier:

Hard Voting:

- In hard voting, the final prediction is made by a majority vote among the individual models. The class that receives the most votes from the ensemble models is selected as the final prediction.
- For classification tasks, hard voting is appropriate when the base models can output class labels.

Soft Voting:

- In soft voting, the final prediction is based on the average of the predicted probabilities (or confidences) from individual models. The class with the highest average probability across all models is selected as the final prediction.
- Soft voting is suitable when the base models can estimate probabilities or confidences for predicted classes.

While running the Voting Classifier, the soft voting was giving the lower accuracy, and this was due to its working algorithm where it provides weights to the predictions made by each of the classifier, whereas while using the hard voting classifier the accuracy achieved was basically that of a simple XG Boost classifier as it was giving the maximum accuracy.

```
from sklearn.ensemble import VotingClassifier
clf1=RandomForestClassifier()
clf2=XGBClassifier()
clf3=DecisionTreeClassifier()
clf4=LinearSVC()
clf5=ExtraTreesClassifier()
model1=VotingClassifier(estimators=[('RFC',clf1),('xgb',clf2),('DT',clf3)]
,voting='soft')
model.fit(X_train,Y_train)
y_pred = model1.predict(X_test)
```

Here the RFC, xgb, DT refers to Random Forest Classifier, XGBoost classifier and decision trees respectively. The voting type shown in the code is of soft type.

Method-3: Stacking Classifier

In ensemble learning, stacking, also known as stacked generalization, is a technique that combines multiple individual models (base learners) with a meta-model (or blender) to make predictions. While stacking is often associated with neural networks in deep learning, it can also be applied to non-neural ensemble methods in traditional machine learning.

Here's how stacking works for non-neural ensemble methods:

1. Base Learners:
   - Similar to other ensemble techniques, stacking starts with a set of diverse base models (e.g., random forests, decision trees, SVMs, etc.). Each base learner is trained on the training data independently.

2. Predictions from Base Models:

- After training, the base models generate predictions for the training data (and sometimes for validation data if cross-validation is used). These predictions serve as new features or meta-features.

3. Meta-Model:
   - Another model, often called the meta-model or blender, is trained on the predictions (meta-features) generated by the base models. This meta-model learns to combine the predictions from the base models to make the final prediction.

4. Making Predictions:
   - During the prediction phase, the trained base models are used to generate predictions for the unseen test data. These predictions are then fed into the trained meta-model, which combines them to make the final prediction.

```
from sklearn. model_selection import train_test_split
from sklearn. metrics import accuracy_score
from sklearn.ensemble import StackingClassifier
models_ensemble[ 'Stacked'] = StackingClassifier(estimators=[('m1',
models['DT']),('m2', models['RF'])], final_estimator=models['XG'])
models_ensemble['Stacked'].fit(X_train,Y_train)
y_pred = models_ensemble['Stacked'].predict(X_test)
print(accuracy_score(Y_test,y_pred))
print(f1_score(Y_test,y_pred))
```

Explanation:

- Stacking Classifier aggregates predictions from base models (models['DT'] and models['RF']) and uses the predictions as features to train the final meta-estimator (models['XG']). This process allows the final model to learn how to best combine the predictions from the base models to make the ultimate prediction.
  (RF - Random Forest, DT - Decision Trees, XG - Xtreme Gradient Boosting)
- The stacked model attempts to leverage the strengths of multiple algorithms (Decision Tree and Random Forest in this case) and their diverse predictions to improve overall performance.

Section - 3 Neural Networks

Neural Networks consists of layers. Each layer consists of neurons. There are 3 types of layers: input layers, hidden layers and output layers.Each connection between neuron has an associated weight, and network learns by adjusting these weights based on training data.

In preprocessing, Standard Scaler was applied for feature scaling, category encoder was used to handle categorical data and outlier removal techniques were employed.

Keras Tuner was utilized for hyperparameter tuning. Keras Tuner is a hyperparameter tuning library for Keras. Keras Tuner is designed to automatically search for the optimal hyperparameters for a given neural network model. Keras Tuner systematically explores the hyperparameter space to find the configuration that yields the best results.

Our **best model** had 6 layers. Layer1 had 64 neurons, Layer2 had 128 neurons, Layer3 had 64 neurons, Layer4 had 32 neurons, Layer5 had 16 neurons. Hidden layers used relu activation function. Since, its a binary classification problem the output layer had 1 neuron with a sigmoid activation function. It gave an accuracy of 0.65481 in kaggle

We tried out methods like dropout(regularization technique to prevent overfitting) , early stopping(regularization technique to prevent overfitting), increasing epochs (helps model to learn more complex patterns from the data), changing the number of neurons, changing the number of layers, and adjusting the learning rate, but the accuracy didn't improve.

Neural networks generally yield good results; however, due to the limited number of columns in our dataset the accuracy didn't improve.

<u>Submission Logs:</u>

1. SVM: Linear kernel (with C=1) gives an accuracy of 0.55425
2. SVM: Polynomial kernel (degree = 3, C=1) gives an accuracy of 0.55542
3. SVM: RBF kernel (with C = 1) gives accuracy of 0.55573
4. Soft Voting with Decision Trees, Random Forest, and XG Boost Classifier gave an accuracy of 0.66721
5. Soft Voting with Decision Trees, Random Forest, XG Boost and Extra Trees Classifier gave an accuracy of 0.62
6. Hard voting with Decision Trees, Random Forest, XG Boost and Extra Trees Classifier gave an accuracy of 0.6435
7. Bagging with XGBoost with n_estimators=20 and random state = 0, gave an accuracy of 0.66823
8. Bagging with XGBoost with n_estimators=20 and random state = 23, gave an accuracy of 0.66856

9. Stacking classifier with base models Random Forest and Decision Trees and the meta estimator as XG Boost gave an accuracy of 0.66375, several different variations were tried and this was the best accuracy using Stacking Classifier.
10. Neural Networks: 6 layers(64-relu,128-relu,64-relu,32-relu,16-relu, 1-sigmoid) with min-max scaler, batch-normalization and learning rate=0.001 gave an accuracy of 0.65396.
11. Neural Networks: 6 layers(64- leaky relu, 128- leaky relu,64- leaky relu,32- leaky relu,16-leaky relu, 1-sigmoid) with standard scaler, batch-normalization and learning rate=0.01 gave an accuracy of 0.65323.
12. Neural Networks: 6 layers(64-relu,128-relu,64-relu,32-relu,16-relu, 1-sigmoid) with standard scaler, batch-normalization and learning rate=0.001 gave an accuracy of 0.65275.
13. Neural Networks: 7 layers(64-relu,128-relu,64-relu,64-relu, 32-relu,16-relu, 1-sigmoid) with standard scaler and learning rate=0.001 gave an accuracy of 0.65241.
14. Neural Networks: 6 layers(64-leaky relu,128-leaky relu,64-leaky relu,32- leaky relu,16-leaky relu, 1-sigmoid) with standard scaler and learning rate=0.01 gave an accuracy of 0.65258.
15. Neural Networks: 5 layers(64-relu,128-relu,64-relu,32-relu,1-sigmoid) with standard scaler and learning rate=0.001 gave an accuracy of 0.65068.

<u>**Best Submission Stats:(Accuracy = 0.66921)**</u>

- Ensemble Technique: Bagging Classifier
- Base Estimator: XGBoost Classifier
- Base Estimator Configuration:
    - Number of Estimators: 20
    - XGBoost Hyperparameters:
        - n_estimators**: 280**
        - max_depth**: 9**
        - learning_rate**: 0.1**
        - min_child_weight**: 1**
        - subsample**: 0.8**
        - colsample_bytree**: 0.8**
        - gamma**: 0**
        - reg_alpha**: 0.01**
        - reg_lambda**: 1**
        - objective**: 'binary:logistic'**
- Utilized the augmented training dataset(augmented_train.csv)
- No feature was dropped

- No filtering of the values was applied on any feature like removal of outliers and others, as when experimented led to a decrease in accuracy.

Link to the competition:
https://www.kaggle.com/competitions/bus-on-time-or-delayed-eval-2