Final Report Churn Rate Prediction Challenge

The goal of this analysis is to accurately predict whether a customer will churn or not. In this regard a training dataset of 4000 rows 18 columns (one of them the response variable is provided). The analysis will be structured as follows. Firstly, some general insights on the data will be provided together with some descriptive statistics and plots aiming to identify potential non-linear relationships in the data. Secondly, a thorough walk-through of the analysis, including the different models and the reasoning for utilisation of these models, as well as the different hypertuning steps will be provided. Finally, model stacking techniques are discussed and utilised in order to optimize the final outcome of the analysis.

Section 1: Preliminary Cleaning and Analysis

Subsection 1A: Cleaning

As outlined before, the data comprises 18 columns and 4000 rows, whereas the training set contains 1986 rows. A first look at the data reveals that most of the columns seem to consist of categorical columns of maximum 4 different categories. The categorical variables included demographic information, contract information and billing and payment information. A deeper analysis into related columns, e.g. phone & lines, as well as Internet & the various options have a complete overlap when people opted to not choose that particular service. In fact, the phone column can be perfectly explained through lines and hence can be dropped as a variable for consideration. The table below further highlights the high collinearity between the different internet options.

Figure 1:

TV and Movies packages grouped and counted

TV	movies	
No	No	1169
	Yes	455
No internet service	No internet service	833
Yes	No	434
	Yes	1109

dtype: int64

The high collinearity of much of the categorical columns is something which will need to be considered when building the models later used for classification. Several of the common models utilised, including Logistic Regression or a Neural Network architecture require the removal of perfect or near perfect multicollinearity from the data. As there are several techniques to deal with this issue, the selection of the categorical features will be handled on a model-by-model basis.

Furthermore, the dataset consists of three non-categorical/binary variables - Time,ChargesMonth and ChargesTotal. A quick look at the correlation matrix (the matrix also includes numerical dummies) reveals a high correlation between the ChargesTotal column and ChargesMonth and Time. This is unsurprising given that the ChargesTotal is simply a collection of all the monthly payments over the history of the customer. It is therefore not useful to consider all three columns. Instead it can be observed that when the ChargesTotal are above the ChargesMonth value, a customer has been paying more for his contract recently than before and vice versa. Hence, instead of using the ChargesTotal column, a new input variable was developed, calculating the Premium or Discount the customer is currently paying over the average monthly payment. This decorrelates the

variables from each other whilst enabling to keep all the information contained in the variables. Finally, the ChargesTotal column contains NaN values. The null values here indicate a new customer, as observed through the fact that they have time = 0. As it is highly unlikely a new customer can actually be recorded as churned in its first month, a dummy is created suchthat models like the logistic regression may actually capture the effect of such customers. The correlation matrices below show both the correlations before and after the variable transformation was carried out.

Figure 2:

	y	senior	time	chargesMonth	chargesTotal
y	1.000000	0.152268	-0.358064	0.186510	-0.209147
senior	0.152268	1.000000	0.008817	0.217438	0.093828
time	-0.358064	0.008817	1.000000	0.262295	0.833457
chargesMonth	0.186510	0.217438	0.262295	1.000000	0.652975
chargesTotal	-0.209147	0.093828	0.833457	0.652975	1.000000

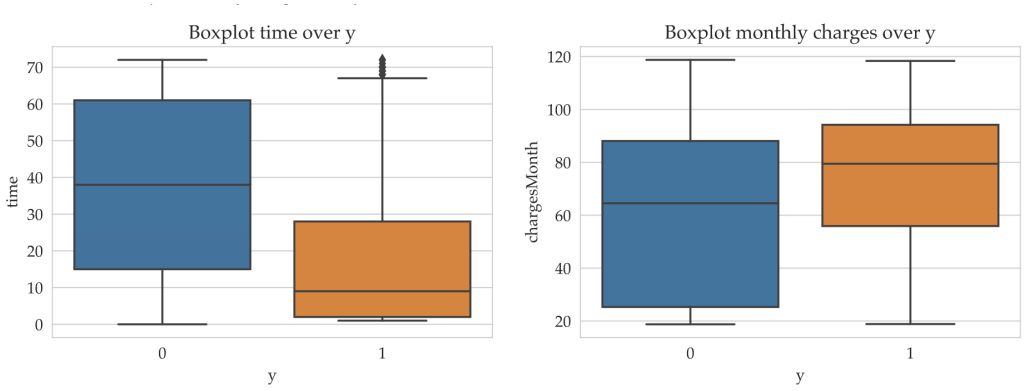
	y	senior	time	chargesMonth	new_customer	month_vs_avg
y	1.000000	0.152268	-0.358064	0.186510	-0.025141	0.008145
senior	0.152268	1.000000	0.008817	0.217438	-0.018629	0.011467
time	-0.358064	0.008817	1.000000	0.262295	-0.055265	-0.025569
chargesMonth	0.186510	0.217438	0.262295	1.000000	-0.045850	-0.008899
new_customer	-0.025141	-0.018629	-0.055265	-0.045850	1.000000	-0.001438
month_vs_avg	0.008145	0.011467	-0.025569	-0.008899	-0.001438	1.000000

Subsection 1B: Descriptives

In this subsection, a somewhat more detailed examination of the data is carried out. In particular, a brief insight on the churn rate across different columns will be given. Next, several non-linear relationships, such as interactions will be explored.

The first thing to be evaluated are the distribution of churn rates across the different categorical columns. It can be quite quickly observed that there is quite large differences in the average churn rate for customers of the different groups. In particular, it seems that customers who have more extensive support and security are much less likely to unsubscribe. It is somewhat surprising that at the same time it seems that customer who have no internet package are also less likely to unsubscribe. Whats more is that customers who have the Fiber optic internet plan have a very high churn rate. This hints towards the fact that customers might be unsatisfied with the internet options provided overall and in particular the fiber optic plan. Similar findings can be confirmed when using boxplots to understand the relationship between the continuous variables and y.

Figure 3 - Boxplots across response variable:



Finally, some interactions were investigated, in particular between some of the continuous variables and the categorical variables. Violin Plots were utilised in order to investigate a different distribution of Y across two columns. An example plotting the distribution of Y across all the different continuous variables and internet options can be found in the code. As can be observed, it does seem indeed that the distributions of Y are changing across such column in a non-linear way. Some early

analysis however on capturing interaction effects beyond taking the polynomials of the continuous regressors showed no significant improvement in the results and was hence discarded from further discussion in this analysis.

Section 2: Analysis

The analysis of each of the models follows a similar pattern throughout. The list of steps carried out for each of the models:

1. Preprocessing of the numerical columns (where necessary)
2. Preprocessing of the categorical columns (i.e. One Hot Encoding or Ordinal Encoding)
3. Hypertuning including changing the probability threshold for each of the different models - using 5-fold Stratified cross-validation
4. Predicting the y_hats according to the test_data inputs

The sklearn pipeline is utilized heavily throughout the analysis to properly define each of the steps to finally fit the model and for utilising a GridSearch to hypertune the parameters later on. As this analysis is both hypertuning the parameters and optimising the probability threshold jointly, a new GridSearch class was created, effectively providing similar features and identical calling method to the GridSearchCV class of sklearn, whilst allowing to provide the best scores across a range of different probability thresholds chosen. The probability threshold is tested always from 0.2-0.9 including (steps of 0.1). The code is not provided here but, I can send it per email if requested.

Whenever possible, weights are inserted into the loss function to directly incorporate the additional focus on correctly predicting customers who churn into the training of the model. In such cases it is expected that the optimal probability threshold remains at 0.5 although this might be varying. As a means of scoring the optimal parameters in hypertuning, a function is created that computes the weighted misclassification error as defined in the instructions.

Subsection 2.1: Bagging and Random Forest

As initial models, some of the common ensemble methods are considered. While it could be opted to choose some of the more "simple" models such as logistic regression first, these ensemble methods have the benefit of needing little to no variable selection or preprocessing. Due to their composition, i.e. through averaging together multiple decision trees based on bootstrapping on the sample and (in case of the random forest) the features, they tend to not suffer from the overfitting problem. Hence, the bagging/random forest provides a good initial overview of the approximate ballpark of expected error, without the need for testing around with different feature selection methods.

The model to be ensembled is chosen to be the DecisionTreeClassifier. Decision Trees work particularly well with these ensemble methods as decision trees are able to build highly complex models, based on which models such as the RandomForest are then able to reduce the variance through their bootstrap mechanism. In particular the Random Forest is expected to have superior performance over Bagging in its ability to decorrelate through bootstrapping on the feature selection.

For both the bagging and the random forest, the number of estimators (i.e. tree models fitted) is an important tuning parameter. Furthermore the depth of the tree is controlled, where generally

more complex trees should perform better. Finally in the random forest, the number of features to bootstrap needs to be selected. The results for the random forest are summarised below:

Figure 4 - Random Forest Parameters:

```
Parameter model__n_estimators: 200
Parameter model__max_features: log2
Parameter model__max_depth: 5
Probability 0.5
Mean cross-validation error: 363.93333333333334, std 27.52324270301174
```

As expected, the Random Forest performs better than the Bagging classifier with a best 5-fold cross-validation score of 361. It seems that the optimal maximum depth for the models lies at a lower level, implying that less complex trees are providing better results. This can potentially arise from the many overlapping features, meaning that a lower depth of the tree can capture already a large part of the bias. Subsequently, the two models will be compared to the performance of other ensemble methods such as boosting.

Subsection 2.2. Boosting

Boosting methods is a further ensemble tool which however follow a very different intuition. Rather than aggregating together different complex trees, boosting instead relies on adaptively learning on a sequency of very flexible models. The boosting algorithm initially fits a tree of for example depth one and computes the residuals. Another tree is subsequently fitted with adjusted weights on the missclasified inputs in order to focus on classifying these observations. This procedure is continued for a number of steps that is defined. Furthermore through setting a learning rate, the analysis can be constructed more and more locally to learn in an optimal way on the datapoints.

In this analysis both AdaBoosting and Gradient Boosting will be carried out. As mentioned before, both the learning rate as well as the number of estimators need to be hypertuned, in particular they follow an inverse relationship with eachother. The more localized we want to estimate our model, the more estimation run it will require. Finally, the tree depth is controlled for options between 1-3.

Figure 5 - Ada Boost Parameters

```
Parameter model__n_estimators: 300
Parameter model__learning_rate: 0.06
Parameter model__base_estimator__max_depth: 1
Probability 0.5000000000000001
Mean cross-validation error: 343.6, std 14.277254638059796
```

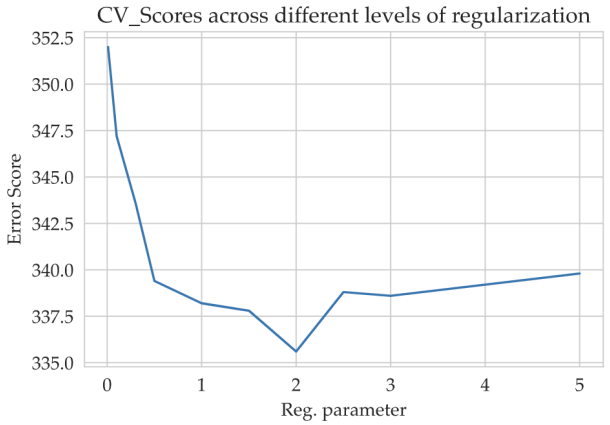
We can clearly see an improvement in the Cross-validation error as compared to the random forest. It seems indeed in this case the data is better understood through the boosting approach where the performance is around 17 points better. Among the two boosting methods, the performance is very similar, the gradient boosting algorithm performs slightly better with a gain of 2 points.

Subsection 2.3 Logistic Regression

The fact that there are a lot of categorical values in the data is something that should be considered in more depth. In particular, it is likely that the general gain from ensemble methods over some of the more traditional classifiers is limited due to the fact that including dummy terms in the regression are essentially able to capture the same localization as a tree is for such variables. Hence the logistic regression is applied in order to understand how it compares to the ensemble methods in this particular case. A number of preprocessing steps need to be made. Firstly, the categorical

columns are One Hot encoded, where strong regularization will serve as the lever to remove the multicollinearity of these created columns. For the numerical columns, different polynomial degrees are tested and subsequently standardized using the standard scaler. Finally, lambda, the level of l2 regularization applied needs to be tuned. Hypertuning showed an optimal polynomial degree of 3, with a regularization parameter of 2. The results are plotted over the different tuning of the regularization parameter.

Figure 6 - Cross-validation scores for Logistic Regression



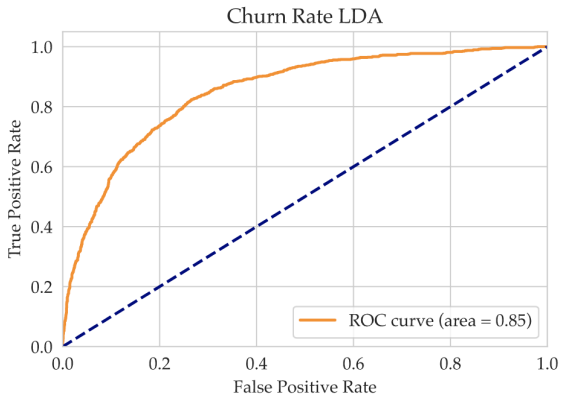
Indeed, it can be confirmed that the performance of the logistic regression seems to be superior in this case. With a cross-validation error of 235, the model is outperforming all the models considered up to now. It seems that beyond the localization through categorical variables, the data can be quite well explained through a function approximation of the continuous variables with a polynomial degree of 3.

Subsection 2.4 Discriminant Analysis

Discriminant Analysis in this case might be a particularly appropriate in modeling due to its ability to well deal with different weights for the different label through selecting the labels based on the probability threshold as well as the ability to include priors into the analysis. In particular, the prior probabilities which are utilised in the calculation of the decision boundary in combination with the likelihood will be adjusted thereby shifting the decision boundary in favour of correctly classifying the customer who churn. Feature selection is performed using a Decision Tree Classifier, which by providing the importance weights can provide a fast and efficient way of understanding the important features. Otherwise similar preprocessing steps as for the logistic regression are utilised. For the optimal parameter selection the ROC curve is shown to observe the effect over the different probabilities.

As can be observed, the results of the Linear Discriminant analysis are very promising. On par with the cross validation results of the logistic regression an LDA with priors of (0.32,0.68) is performing very strongly.

Figure 7 - ROC Curve



Subsection 2.5 Feed Forward Neural Networks - The Multilayer Perceptron (MLP):

The final model considered in further detail for this analysis is the multilayer Perceptron, which as a univeral function approximator might be able to best understand the relationship between the data. Given the successes of the Logistic Regression, it might be expected that a smaller number of hidden layers can prove to accurately estimate the function that can classify y. There are a number of tuning parameters to consider. Namely, the number of epochs (i.e. training runs), batch size (i.e. sample size seen during a training run), size and number of hidden dimension and the learning rate.

As can be seen in the results, while the performance of the multilayer perceptron seems to be quite good it does not manage to compete with the Logistic Regression or LDA results.

Further models have been considered such as Support Vector Machines or using MARS to smartly create and select the features to be used later in classification, however as they did not provide any improvement they are not further discussed.

Section 3: Model Stacking and Conclusion

Stacking and Bayesian Model Averaging are further ensemble techniques which aim to combine the predictions of different models together in order to obtain one superior performance. Model stacking in particular applies logistic regression on metadata created through LOOCV using the predicted probabilites of the different estimators utilised. While model stacking would be the preferred way of combining the models, some issues arose due to the fact that the hypertuning included setting the optimal probability threshold. The stacking step should have been included in this hypertuning from the beginning, setting the optimal threshold for the stacked combination of the model and understanding through this the optimal combination of parameters and models. As this proved very time costly, it was decided to use a decision rule for combining the models instead.

In particular 4 models were chosen to provide an optimal trade-off between strong accuracy and correlation, which should thereby be improving the results quite well. Namely, the Random Forest, LDA, Logit and Ada Boost. While the random forest performed somewhat worse than the other model, it is seen to provide a good balance with a correlation of 82% to the other models, compared to an average correlation of 90% otherwise. Finally it was decided it would suffice to have 2 models predicting that a customer will churn, whereas at least three models needed to be aligned in order to classify the datapoint to a customer who remains. These predictions were finally uploaded, achieving a misclassification error of 219.

In [1]:
```
%matplotlib inline
%config InlineBackend.figure_format = 'svg'

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import RepeatedStratifiedKFold,GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, GradientBoostingClassifier, BaggingClassifier
from sklearn.preprocessing import LabelEncoder,OrdinalEncoder
from sklearn.metrics import accuracy_score,confusion_matrix,make_scorer
from sklearn.tree import DecisionTreeClassifier
from sklearn.compose import ColumnTransformer,TransformedTargetRegressor
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import RepeatedKFold,StratifiedKFold
from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import OneHotEncoder
from GridCV import Hypertuner_proba

sns.set_style("whitegrid")
plt.rcParams.update({'font.size': 12, 'font.style': 'normal', 'font.family':'palatino'})
```

In [2]:
```
data = pd.read_csv("Data/train.csv")
data["y"] = np.where(data["y"] == 1,0,1)
#data.head()
```

In [3]:
```
test_data = pd.read_csv("Data/test.csv")
#test_data
```

In [4]:
```
corrs = data.corr()
#corrs.style.background_gradient(cmap = "coolwarm")
```

In [5]:
```
#Checking for na-values
#data.isna().sum()
```

In [7]:
```
#Check for collinearity of some of the categorial columns

#print("TV and Movies packages grouped and counted")
#print("-"*50)
#data.groupby(["TV","movies"]).size()
```

In [8]:
```
#Creating the new fields discussed in the report.

data["new_customer"] = np.where(data["time"] == 0,1,0)
test_data["new_customer"] = np.where(test_data["time"] == 0,1,0)

data["avg_charges"] = data["chargesTotal"]/data["time"]
test_data["avg_charges"] = test_data["chargesTotal"]/test_data["time"]

data.loc[data["avg_charges"].isna(),"avg_charges"] = data["chargesMonth"]
test_data.loc[test_data["avg_charges"].isna(),"avg_charges"] = test_data["chargesMonth"]

data["month_vs_avg"] = data["chargesMonth"]/data["avg_charges"]
test_data["month_vs_avg"] = test_data["chargesMonth"]/test_data["avg_charges"]
```

In [9]:
```
data = data.drop(["avg_charges","chargesTotal"],axis = 1)
test_data = test_data.drop(["avg_charges","chargesTotal"],axis = 1)
```

In [10]:
```
corrs = data.corr()
#corrs.style.background_gradient(cmap = "coolwarm")
```

In [11]:
```
#This prints for each of the categorical values the share of customers who churn

"""categorical_columns = list(data.columns[data.dtypes == object])

categorical_columns.append("senior")
for col in categorical_columns:
    print()
    print(f"Churn rate grouped by {col}")
    print("-"*30)
    print(data.groupby(col).agg({"y":["mean","count"]}))"""

categorical_columns = list(data.columns[data.dtypes == object])
num_cols = ['time', 'chargesMonth', 'month_vs_avg']
```

In [12]:
```
#Phone is the higher level of lines and is hence perfectly multicollinear with it. It can be dropped.

data = data.drop("phone",axis = 1)
test_data = test_data.drop("phone",axis = 1)
```

In [13]:
```
#Violin plots for interaction checking

"""fig,axs = plt.subplots(1,3,figsize = (16,6))

for i,ax in enumerate(fig.axes):
    ax.set_title(f"Distribution of {num_cols[i]}")
    sns.violinplot(x = "internet", y = num_cols[i],hue = "y",data = data,ax = ax,split=True)"""
```

Out[13]:
```
'fig,axs = plt.subplots(1,3,figsize = (16,6))\n\nfor i,ax in enumerate(fig.axes):\n    ax.set_title(f"Distribution of {num_cols[i]}")\n    sns.violinplot(x = "in
ternet", y = num_cols[i],hue = "y",data = data,ax = ax,split=True)'
```

In [14]:
```
#Boxplots of the continuouse variables across y

"""fig, axs = plt.subplots(1,2, figsize = (13,4))

ax = axs[0]
sns.boxplot(x = "y",y = "time",data = data,ax = ax)
ax.set_title("Boxplot time over y")

ax = axs[1]
sns.boxplot(x = "y",y = "chargesMonth",data = data,ax = ax)
ax.set_title("Boxplot monthly charges over y")"""
```

Out[14]:
```
'fig, axs = plt.subplots(1,2, figsize = (13,4))\n\nax = axs[0]\nsns.boxplot(x = "y",y = "time",data = data,ax = ax)\nax.set_title("Boxplot time over y")\n\nax =
axs[1]\nsns.boxplot(x = "y",y = "chargesMonth",data = data,ax = ax)\nax.set_title("Boxplot monthly charges over y")\n'
```

In [16]:
```
#Lets first make a random forest classifier
X = data.iloc[:,1:]
y = data["y"]
categorical_columns = X.columns[X.dtypes == object]
cv = StratifiedKFold(5)
```

In [17]:
```
#Creating the function for the weighted accuracy

def weighted_accuracy_score(y, y_hat,**kwargs):
    confusion = confusion_matrix(y,y_hat)
    valueTotal = confusion[1,0]*5+confusion[0,1]*1
    return valueTotal
```

Bagging

In [18]:
```
from sklearn.ensemble import BaggingClassifier

categorical_cols = OrdinalEncoder()

column_transformer_bag = ColumnTransformer([
    ("cat_trans", categorical_cols, categorical_columns)], remainder = "passthrough")

pipeline_bag = Pipeline(steps = [
    ("transformer", column_transformer_bag),
    ("model", BaggingClassifier(DecisionTreeClassifier(class_weight = {0: 1, 1: 5}),n_jobs = -1))
])

parameters_bag = {"model__n_estimators": [200],
              "model__base_estimator__min_samples_split": [2,4,6,8,16],
              "model__base_estimator__min_samples_leaf": [1,2,3,4,8]
             }
#search_Bagging = Hypterunter_proba(pipeline,parameters,scoring = weighted_accuracy_score,prob_range = np.arange(0.1,1,0.1) )
#search_Bagging.fit(X,y)
#search_Bagging.best_params,search_Bagging.best_mean_score,search_Bagging.best_prob
```

In [19]:
```

```

Random Forest

In [72]:
```
categorical_col_RF = OrdinalEncoder()

column_transformer_RF = ColumnTransformer([
    ("cat_trans", categorical_col_RF, categorical_columns)], remainder = "passthrough")

pipeline_RF = Pipeline(steps = [
    ("transformer", column_transformer_RF),
    ("model", RandomForestClassifier(class_weight = {0: 1, 1: 5},n_jobs = -1))
])

parameters_RF = {"model__n_estimators": [200],
              "model__max_features": [0.2,0.5,"sqrt","log2"],
              "model__max_depth" : [None,5,10]
             }
#search_RF = Hypertuner_proba(pipeline_RF,parameters_RF,scoring = weighted_accuracy_score,prob_range = np.arange(0.1,1,0.1) )
#search_RF.fit(X,y)
```

In [29]:
```
def printer(search):
    for key,val in search.best_params.items():
        print(f"Parameter {key}: {val}")
    print(f"Probability {search.best_prob}")
    print(f"Mean cross-validation error: {search.best_mean_score}, std {search.best_cv_scores_std}")
#printer(search_RF)
search_RF.best_mean_score,search_RF.best_cv_scores_std
```

Out[29]:
```
(363.93333333333334, 27.52324270301174)
```

In [31]:
```
pipeline_RF.set_params(**search_RF.best_params)

pipeline_RF.fit(X,y)
y_hat_RF = np.where(pipeline_RF.predict_proba(test_data)[:,1] >= 0.5,1,0)+1
```

Adaboosting

In [73]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import OneHotEncoder


categorical_cols = OneHotEncoder()

column_transformer_ada = ColumnTransformer([
    ("cat_trans", categorical_cols, categorical_columns)], remainder = "passthrough")

pipeline_ada = Pipeline(steps = [
    ("transformer", column_transformer_ada),
    ("model", AdaBoostClassifier(DecisionTreeClassifier(class_weight={0:1,1:5})))
])


parameters_ada = {"model__n_estimators": [300,400,500],
              "model__learning_rate": [0.04,0.05,0.06],
              "model__base_estimator__max_depth": [1],
             }

#search_Ada = Hypertuner_proba(pipeline_ada,parameters_ada,scoring = weighted_accuracy_score,prob_range = np.arange(0.2,0.9,0.1),cv = StratifiedKFold(n_splits=5))
#search_Ada.fit(X,y)
#printer(search_Ada)
```

In [35]:
```
pipeline_ada.set_params(**search_Ada.best_params)
pipeline_ada.fit(X,y)
y_hat_Ada = np.where(pipeline_ada.predict_proba(test_data)[:,1] >= 0.5,1,0)+1
```

Gradient Boosting Classifier

In [74]:
```
from sklearn.ensemble import GradientBoostingClassifier

categorical_cols = OrdinalEncoder()

column_transformer_XGBoost = ColumnTransformer([
    ("cat_trans", categorical_cols, categorical_columns)], remainder = "passthrough")

pipeline_XGBoost = Pipeline(steps = [
    ("transformer", column_transformer_XGBoost),
    ("model", GradientBoostingClassifier())
])

parameters_ada = {"model__n_estimators": [400,500],
              "model__learning_rate": [0.05,0.1],
              "model__subsample": [0.5,0.6],
              "model__max_depth": [1]
             }

#search_GBoost = Hypertuner_proba(pipeline_XGBoost,parameters_ada,scoring = weighted_accuracy_score,prob_range = np.arange(0.2,0.9,0.1),cv = StratifiedKFold(n_splits=5))
#search_GBoost.fit(X,y)
#printer(search_GBoost)
```

Logistic Regression

In [52]:
```
from sklearn.preprocessing import OneHotEncoder
from GridCV import Hypertuner_proba

categorical_cols = Pipeline(steps = [
    ("encode", OneHotEncoder(handle_unknown="ignore"))

])

num_transformer_logit  = Pipeline(steps = [
    ("polynomial", PolynomialFeatures()),
    ("scale", StandardScaler())
])

column_transformer_logit = ColumnTransformer([
    ("cat_trans", categorical_cols, categorical_columns),
    ("num_trans", num_transformer_logit, num_cols)
],remainder = "passthrough")


pipeline_logit = Pipeline(steps = [
    ("transformer", column_transformer_logit),
    ("model", LogisticRegression(penalty = "l2",class_weight = {0:1,1:5},solver = "lbfgs",max_iter=2000))
])


parameters_logit = {"model__C": [0.01,0.1,0.3,0.5,1,1.5,2,2.5,3,5],
              "transformer__num_trans__polynomial__degree": [3]
             }


#search_logit = Hypertuner_proba(pipeline_logit,parameters_logit,scoring = weighted_accuracy_score,prob_range = np.arange(0.2,0.9,0.1),cv = StratifiedKFold(n_splits=5))
#search_logit.fit(X,y)
#printer(search_logit)
```

In [55]:
```
#Plot for the error scores

"""results_logit = search_logit.grid_search_results

x = results_logit.loc[results_logit["prob"] == search_logit.best_prob]

plt.plot(x["params"].apply(pd.Series)["model__C"],x["mean_score"])
plt.title("CV_Scores across different levels of regularization")
plt.xlabel("Reg. parameter")
plt.ylabel("Error Score")"""
```

Out[55]:
```
'results_logit = search_logit.grid_search_results\n\nx = results_logit.loc[results_logit["prob"] == search_logit.best_prob]\n\nplt.plot(x["params"].apply(pd.Series)["model__C"],x["mean_score"])\nplt.title("CV_Scores across different levels of regularization")\nplt.xlabel("Reg. parameter")\nplt.ylabel("Error Score")'
```

In [58]:
```
pipeline_logit.set_params(**search_logit.best_params)
pipeline_logit.fit(X,y)
y_hat_Logit = np.where(pipeline_logit.predict_proba(test_data)[:,1] > search_logit.best_prob,1,0)+1
```

LDA

In [76]:
```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis,QuadraticDiscriminantAnalysis
from sklearn.feature_selection import SelectKBest,chi2,SelectFromModel
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import StratifiedKFold

categorical_cols = OneHotEncoder(handle_unknown="ignore")

num_transformer_LDA  = Pipeline(steps = [
    ("polynomial", PolynomialFeatures()),
    ("scale", StandardScaler())
])

column_transformer_LDA = ColumnTransformer([
    ("cat_trans", categorical_cols, categorical_columns),
    ("num_trans", num_transformer_LDA, num_cols)

],remainder = "passthrough")


pipeline_LDA = Pipeline(steps = [
    ("transformer", column_transformer_LDA),
    ("selector",SelectFromModel(DecisionTreeClassifier(class_weight = {0:1,1:5}))),
    ("model", LinearDiscriminantAnalysis())
])

parameters_LDA = {
             "selector__threshold": ["0*mean","0.001*mean","0.002*mean"],
             "transformer__num_trans__polynomial__degree" : [3],
             "model__priors" : [[0.32,0.68],[0.3,0.7],[0.4,0.6],[0.5,0.5]],
}

#search_LDA = Hypertuner_proba(pipeline_LDA,parameters_LDA,scoring = weighted_accuracy_score,prob_range = np.arange(0.2,0.9,0.1),cv = StratifiedKFold(n_splits=5))
#search_LDA.fit(X,y)
#printer(search_LDA)
```

In [64]:
```
pipeline_LDA.set_params(**search_LDA.best_params)
pipeline_LDA.fit(X,y)
probs_train = pipeline_LDA.predict_proba(X)
y_hat_LDA = np.where(pipeline_LDA.predict_proba(test_data)[:,1] > 0.5,1,0)+1
```

In [77]:
```
from sklearn.metrics import roc_curve, auc

"""fpr,tpr,_ = roc_curve(y,probs_train[:,1])
roc_auc = auc(fpr, tpr)

plt.figure()
lw = 2
plt.plot(fpr, tpr, color='darkorange',
         lw=lw, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic example')
plt.legend(loc="lower right")
plt.show()
"""
```

Out[77]:
```
'fpr,tpr,_ = roc_curve(y,probs_train[:,1])\nroc_auc = auc(fpr, tpr)\n\nplt.figure()\nlw = 2\nplt.plot(fpr, tpr, color=\'darkorange\',\n         lw=lw, label=\'ROC curve (area = %0.2f)\' % roc_auc)\nplt.plot([0, 1], [0, 1], color=\'navy\', lw=lw, linestyle=\'--\')\nplt.xlim([0.0, 1.0])\nplt.ylim([0.0, 1.05])\nplt.xlabel(\'False Positive Rate\')\nplt.ylabel(\'True Positive Rate\')\nplt.title(\'Receiver operating characteristic example\')\nplt.legend(loc="lower right")\nplt.show()\n'
```

Multilayer Perceptron

In [ ]:
```
#columns to drop when onehot encoding - removes multicollinearity
drop_column = ["Male","No","No phone service","No","No internet service","No internet service","No internet service",
             "No internet service","No internet service","No internet service","No","Mailed check"]
categorical_col_MLP = OneHotEncoder(drop= drop_column)

column_transformer_MLP = ColumnTransformer([
    ("cat_trans", categorical_col_MLP, categorical_columns)], remainder = "passthrough")

pipeline_MLP = Pipeline(steps = [
    ("transformer", column_transformer_MLP),
    ("model", MLPClassifier(max_iter = 500,n_iter_no_change=5,tol = 1e-3)) ])

parameters_MLP = {"model__batch_size": [128,256],
              "model__learning_rate": ["adaptive"],
              "model__learning_rate_init" : [0.0008,0.0005],
              "model__hidden_layer_sizes" : [(100),(128)],
              "model__alpha":[0.0001],
              "model__solver" : ["adam"]
             }
#search_MLP = Hypterunter_proba(pipeline_MLP,parameters_MLP,scoring = weighted_accuracy_score,prob_range = np.arange(0.2,0.9,0.1) )
#search_MLP.fit(X,y)
```

In [24]:
```
#search_MLP.best_params,search_MLP.best_mean_score,search_MLP.best_prob
```

Model Stacking

More was done but not shown due to page limit requirements.

In [78]:
```
models = {"LDA": y_hat_LDA,"Ada": y_hat_Ada,"RF": y_hat_RF, "Logit" : y_hat_Logit}
stacker = pd.DataFrame(models)
#corrs = stacker.corr()
#corrs.style.background_gradient(cmap = "coolwarm")
```

In [70]:
```
#Selction throigh majority vote
y_hat_majority = np.where(stacker.sum(axis = 1) > 5, 2,1)
```

In [71]:
```
with open("Data/final_Majority_Vote_7.txt","w") as file:
    for i in y_hat_majority:
        file.writelines(f"{i}\n")
```