Case Study 8 - XGBoost, Random Forest and SVM

Gabriel Gonzales
DS 7333 - QTW
10/26/2020

Abstract

For this case study, three models were built to determine which had the best accuracy: XGBoost, Random Forest, and a Support Vector Machine(SVM). The data used was anonymous and had no metadata and no data dictionary. It was determined that SVM performed the best.

Introduction

The purpose of this case study was to determine which of the three models would achieve the highest accuracy. Exploration of hypertuning the models was needed to improve the accuracy, log-loss and run time for each model. The following three questions were also given as part of the case study.

1) Build 3 tuned models: An XGBoost, a Random Forest, and an SVM.
2) Show the log loss and accuracy for XGBoost, and Random Forest models on out of fold predictions. Show the Accuracy of the SVM on a validation set
3) Time how long it takes to do a sample of 1000, 2000, 5000, and 10,000 rows in the SVM. What is the rough scaling of SVM with sample size?

Methods

Data

The data given was anonymized, containing both categorical and numerical variables. The original size of the dataset is 114321 rows and 133 columns. There is no missing data, no metadata, no ordinal variables or a data dictionary. The "target" column in the set is the variable to predict.

Exploratory Data Analysis

Through the EDA, there were 19 Categorical and 112 continuous variables within the dataset. Further details of the categorical variables discovered column “v22” had 18,210 unique values; an unusually high number. It was decided to remove this field for its large unique fields. In order to create the models using the categorical variables, each categorical variable was One-Hot encoded.

Feature Reduction

The next step was to reduce the number of highly correlated values. This will improve the time it takes to train the models. The function used to achieve this is called “reduced_features” and is designed to evaluate pairwise correlation and remove the column’s if the correlation is greater than 0.95. Using this method, 38 highly correlated fields were dropped, reducing the number of columns to 438.

Case Study 8 - XGBoost, Random Forest and SVM

Gabriel Gonzales
DS 7333 - QTW
10/26/2020

Question 1: Build three tuned models: XGBoost, Random Forest & SVM

In order to create each model, one must first train and then test each model to evaluate their performance. The log-loss score was also used to determine which parameters have the most influence on a model's performance. The train/ test set for cross-validation of the results was created using the train_test_split function from sklearn's model_selection module with test_size size equal to 30% of the data. A sample of 70% of the data, through a stratified random sample was chosenusing the function random_state 123 in order to reproduce the results. A CV K-Fold of 3 was used in order to minimize the amount of time some of these models takes to run.

XGBoost

The following parameters were selected for tuning our XGBoost model. These parameters were chosen as a "Tree Booster" (1). All the tuning parameters are focused on the boosting aspect of the model.

Hyper-Tuning Parameters

- eta (default=0.3, alias: learning_rate, range: [0,1])
    - Step size shrinkage used in update to prevents overfitting. After each boosting step, we can directly get the weights of new features, and eta shrinks the feature weights to make the boosting process more conservative.
- gamma (default=0, alias: min_split_loss, range: [0,∞])
    - Minimum loss reduction required to make a further partition on a leaf node of the tree. The larger gamma is, the more conservative the algorithm will be.
- max_depth (default=6, range: [0,∞] (0 is only accepted in lossguided growing policy when tree_method is set as hist))
    - Maximum depth of a tree. Increasing this value will make the model more complex and more likely to overfit. 0 is only accepted in lossguided growing policy when tree_method is set as hist and it indicates no limit on depth. Beware that XGBoost aggressively consumes memory when training a deep tree.
- min_child_weight (default=1, range: [0,∞])
    - Minimum sum of instance weight (hessian) needed in a child. If the tree partition step results in a leaf node with the sum of instance weight less than min_child_weight, then the building process will give up further partitioning. In linear regression task, this simply corresponds to minimum number of instances needed to be in each node. The larger min_child_weight is, the more conservative the algorithm will be.

The total number of parameters was 180. The overall time it took to run Xgboost was 1 hour and 29 min. The parameters that had the most influence were gamma, max_depth and eta. Further

analysis showed that decreasing the eta had the largest impact to the performance of the model.

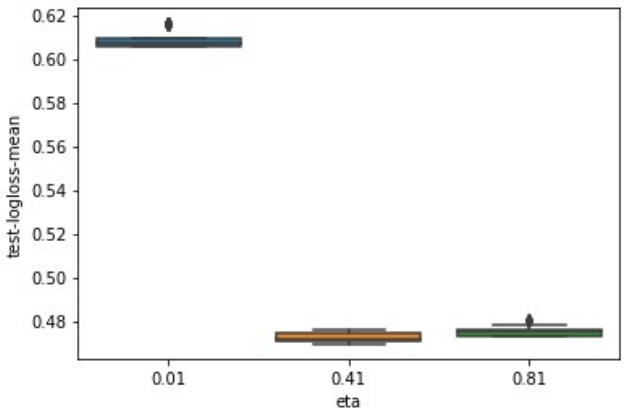


Figure 1. Log-loss-mean score by eta value

Random Forest

The follow parameters selected were used to focus on the tree and leaf features. The definition of the parameters were obtained from the scikit learn website (2).

Hyper-Tuning Parameters

- max_features ({“auto”, “sqrt”, “log2”}, int or float, default=”auto”)
  - The number of features to consider when looking for the best split:
- n_estimators (int, default=100)
  - The number of trees in the forest.
- min_samples_leaf (int or float, default=1)
  - The minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least min_samples_leaf training samples in each of the left and right branches. This may have the effect of smoothing the model, especially in regression.

```
Fitting 3 folds for each of 36 candidates, totalling 108 fits
[Parallel(n_jobs=-2)]: Using backend LokyBackend with 3 concurrent workers.
[Parallel(n_jobs=-2)]: Done  44 tasks      | elapsed:  4.1min
[Parallel(n_jobs=-2)]: Done 108 out of 108 | elapsed: 26.6min finished
```

Figure 2. Random Forest Run Time.

Case Study 8 - XGBoost, Random Forest and SVM

Gabriel Gonzales
DS 7333 - QTW
10/26/2020

The overall time it took to run the Random Forest model was 26.6 min. Setting the parameter of max_features value to either 0.2 or 0.3 produced the best result. Looking at the boxplot below, the parameters for adjust min_samples_leaf produced a more consistent result and high mean score.

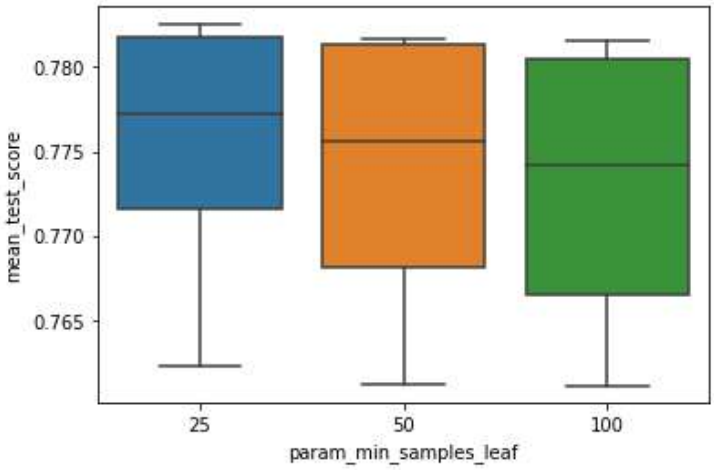


Figure 3. Dist. of mean_test_score by min_samples_leaf

To determine if the accuracy could be further increased, we looked at how Gini and Entropy change accuracy with changing Max Depth of the Trees in Random Forest. Both Gini and Entropy measure impurity. Entropy tells how random of uncertain the dataset is, while Gini provides an index to measure inequality

Both the measurements are represented mathematically below:

$$Gini = 1 - \sum_{i=1}^{n} p^2(c_i)$$

$$Entropy = \sum_{i=1}^{n} -p(c_i)log_2(p(c_i))$$

where $p(c_i)$ is the probability/percentage of class c_i in a node.

Looking at the plots for both Gini and Entropy, the accuracy of the model is are the same for both indices.

Case Study 8 - XGBoost, Random Forest and SVM

Gabriel Gonzales
DS 7333 - QTW
10/26/2020

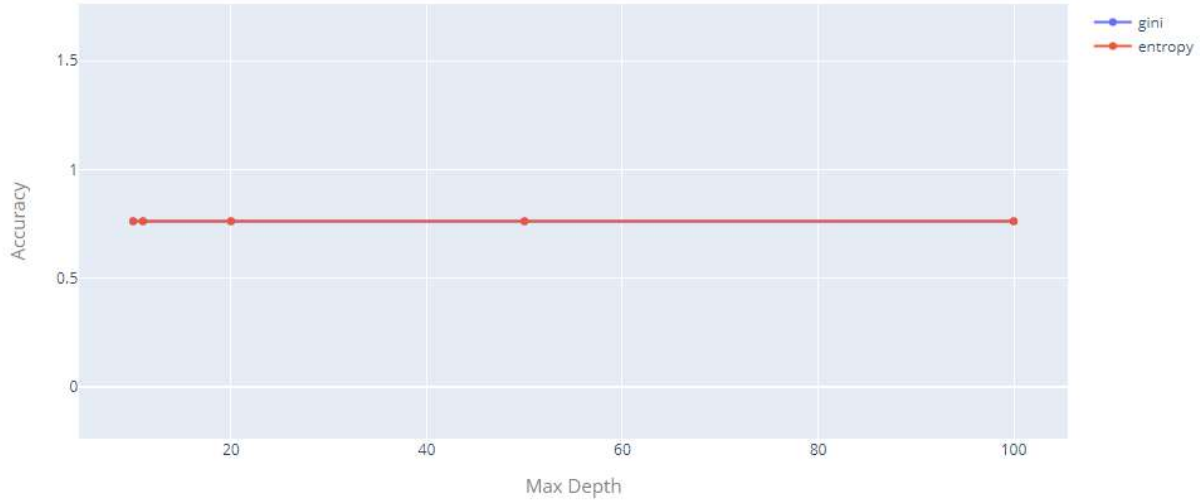


Figure 4. Entropy Plot

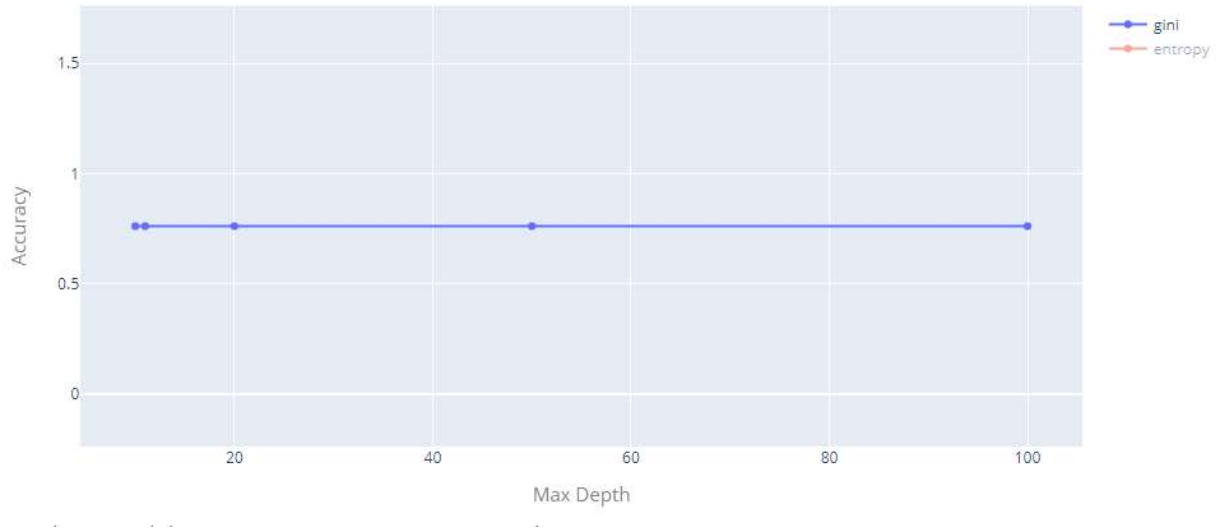


Figure 5. Gini Plot

Support Vector Machine

Some of the benefits of using SVM are that it works relatively well when there is a clear margin of separation between classes, is more effective in high dimensional spaces and is relatively memory efficient. This model will utilize the SKLearn's LinearSVC (3). This is similar to the SVM algorithm, but uses a linear kernel. The change allows more flexibility on penalty and loss function, thus allows us a scale better.Our implementation of SVM uses the StandardScaler which will scale our values between -1 and 1. We chose this in order to allow SVM to consistently measure distance between values.

Case Study 8 - XGBoost, Random Forest and SVM

Gabriel Gonzales
DS 7333 - QTW
10/26/2020

Hyper-Tuning Parameters

- C (float, default=1.0)
    - Regularization parameter. The strength of the regularization is inversely proportional to C. Must be strictly positive.
- loss ({'hinge', 'squared_hinge'}, default='squared_hinge')
    - Specifies the loss function. 'hinge' is the standard SVM loss (used e.g. by the SVC class) while 'squared_hinge' is the square of the hinge loss.
- dual (bool, default=True)
    - Select the algorithm to either solve the dual or primal optimization problem. Prefer dual=False when n_samples > n_features.
- tol (float, default=1e-4)
    - Tolerance for stopping criteria.
- max_iter (int, default=1000)
    - The maximum number of iterations to be run.

```
Fitting 3 folds for each of 36 candidates, totalling 108 fits
```

```
[Parallel(n_jobs=-2)]: Using backend LokyBackend with 3 concurrent workers.
[Parallel(n_jobs=-2)]: Done  44 tasks      | elapsed:  4.3min
[Parallel(n_jobs=-2)]: Done 108 out of 108 | elapsed: 28.5min finished
```

The model performed 108 fits and took 28.5 min to finish.

Question 2: Show the log loss and accuracy for XGBoost, and Random Forest models on out of fold predictions. Show the Accuracy of the SVM on a validation set.

An "out of fold predictions" technique was used to calculate log-loss and accuracy for our 3 models. A stratified k-fold will be performed over 5 iterations. For each hold out in the fold we will predict its value. Below code shows the optimal parameters we have selected for each model.

```
Random Forest {'max_features': 0.3, 'min_samples_leaf': 25, 'n_estimators': 100}
SVM {'max_features': 0.2, 'min_samples_leaf': 25, 'n_estimators': 100}
XGB {'gamma': 12, 'max_depth': 5, 'eta': 0.41, 'min_child_weight': 0.01, 'objective': 'binary:logistic'}
```

Case Study 8 - XGBoost, Random Forest and SVM

Gabriel Gonzales
DS 7333 - QTW
10/26/2020

The plot below indicates where each fold pulls it hold-out value for each k-fold.

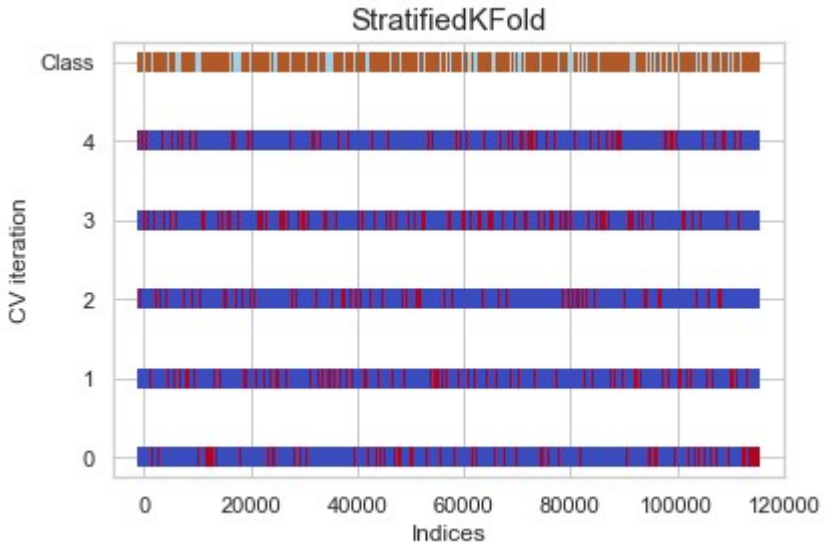


Figure 6. Stratified k-fold

For the final models, it was determined that SVM performed the best with an accuracy of 0.80. Reviewing log loss, it was determined that the Random Forest model performed slightly better.

	metric	XGBoost	Random Forest	Support Vector Machine
0	Accuracy	0.781003	0.781799	0.807944
1	Log Loss	0.472091	0.467973	NaN

Figure 7. Final Results for XGBoost, RF and SVM

Question 3: Time how long it takes to do a sample of 1000, 2000, 5000, and 10,000 rows in the SVM. What is the rough scaling of SVM with sample size?

To determine the relationship between sample size and time for SVM, both variable had to be logged in order to meet the regression assumptions and for the relationship to appear linear. The plot below indicates a linear relationship.

Case Study 8 - XGBoost, Random Forest and SVM

Gabriel Gonzales
DS 7333 - QTW
10/26/2020

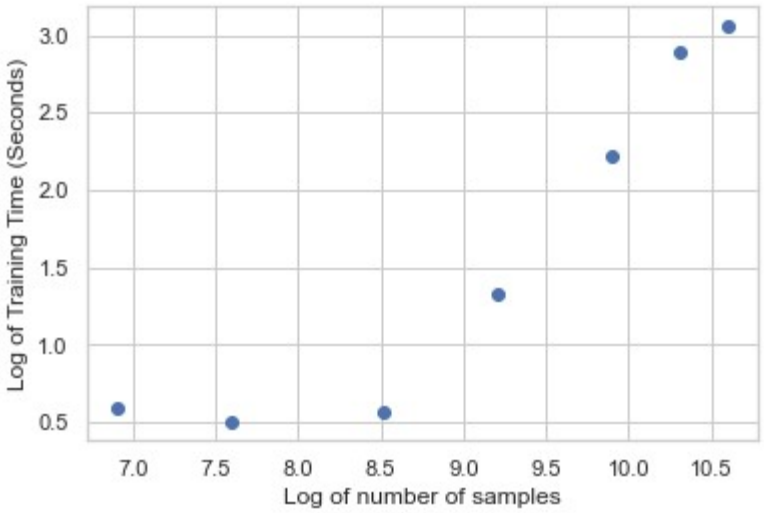
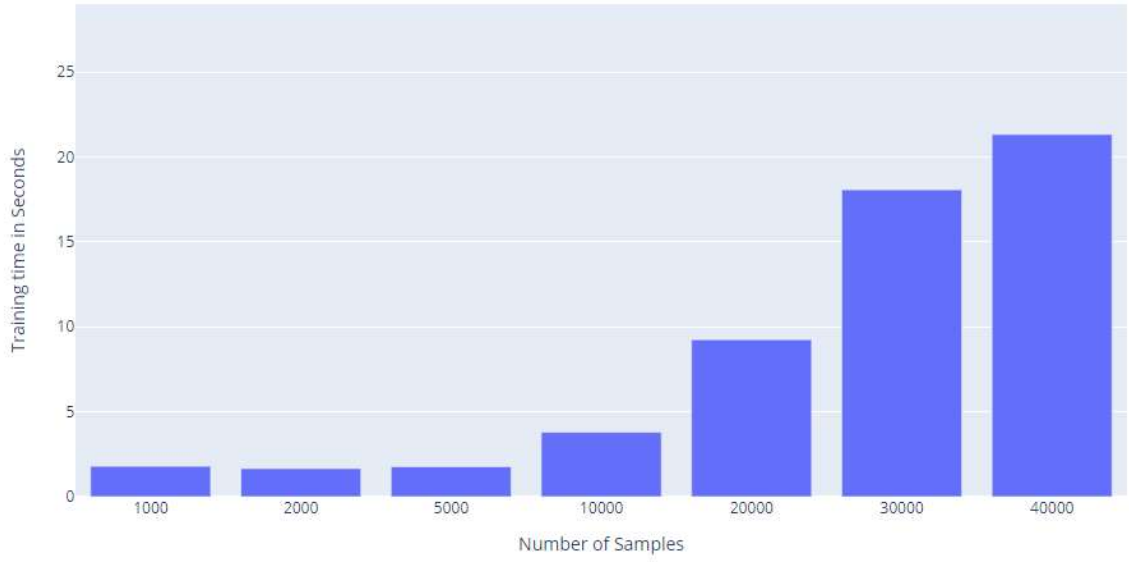


Figure 8. Log scale of time.

The plot below shows the training time per number of samples. As can be seen below, the training time increases exponentially past 10k number of samples.



To determine the rough scaling of SVM with sample size, the Regression (x) coefficient was calculated. Based on the results below, it can be interpreted as for every 1% change in number of samples, the training time for SVM will go up by 0.74%

```
Regression (x) coefficient: 0.7422887007027055
```

Case Study 8 - XGBoost, Random Forest and SVM

Gabriel Gonzales
DS 7333 - QTW
10/26/2020

Conclusion

In this case study three models were tuned: XGBoost, Random Forest, and SVM. The analysis of each model found determined which parameter combinations performed best The analysis also determined that SVM performed the best overall. XGBoost also performed the worst in both Log-Loss and Accuracy. This could be due to low dimensions of columns that would have benefited the boosting techniques.. Future analysis should focus on improving the imbalanced dataset using a technique like SMOTE.

Case Study 8 - XGBoost, Random Forest and SVM

Gabriel Gonzales
DS 7333 - QTW
10/26/2020

References

1. https://xgboost.readthedocs.io/en/latest/parameter.html.
2. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
3. https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html#sklearn.svm.LinearSVC

Appendix – Python Code

```
import numpy as np
def reduce_features(df, verbose = False):
    # calculate the correlation matrix
    corr_matrix  = df.corr().abs()

    # Select upper triangle of correlation matrix
    upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape),
k=1).astype(np.bool))

    # Find index of feature columns with correlation greater than 0.95
    to_drop = [column for column in upper.columns if any(upper[column] >
0.95)]

    #Get all of the correlation values > 95%
    x = np.where(upper > 0.95)

    #Display all field combinations with > 95% correlation
    cf = pd.DataFrame()
    cf['Field1'] = upper.columns[x[1]]
    cf['Field2'] = upper.index[x[0]]

    #Get the correlation values for every field combination. (There must be a
more pythonic way to do this!)
    corr = [0] * len(cf)
    for i in range(0, len(cf)):
        corr[i] =  upper[cf['Field1'][i]][cf['Field2'][i]]

    cf['Correlation'] = corr

    if( verbose ):
        print('There are ', str(len(cf['Field1'])), ' field correlations >
95%.')
        display(cf)
```

Case Study 8 - XGBoost, Random Forest and SVM

Gabriel Gonzales
DS 7333 - QTW
10/26/2020

```
        print('Dropping the following ', str(len(to_drop)), ' highly
correlated fields.')
        to_drop

    #Check columns before drop
    if( verbose ):
        print('\r\n*********Before: Dropping Highly Correlated
Fields************************************')
        display(df.info(verbose=False))

    # Drop the highly correlated features from our training data
    df = df.drop(to_drop, axis=1)

    #Check columns after drop
    if( verbose ):
        print('\r\n*********After: Dropping Highly Correlated
Fields************************************')
        df.info(verbose=False)

    return df
import pickle
import os

case_8_reduced_feats = reduce_features(case_8_final, True)

with open(os.path.join("C:\\Users\\Gabri","case_8_final_feats.pkl"),"wb") as f:
   pickle.dump((y, case_8_reduced_feats), f)
with open(os.path.join("C:\\Users\\Gabri","case_8_final_feats.pkl"),"rb") as f:
   y, case_8_reduced_feats = pickle.load(f)
Next, we will look at the response variables to determine if the data is balanced or not
import matplotlib.pyplot as plt
import seaborn as sns
plt.title('Figure 1: Response Variable', y=-0.25)
plt.xlabel('Class')
plt.ylabel('Frequency')
ax = sns.countplot(x=y)

import pandas as pd

X = case_8_reduced_feats.values

X_train, X_test, y_train, y_test = train_test_split(
      X, y,
      stratify=y,
      test_size=0.30,
```

Case Study 8 - XGBoost, Random Forest and SVM

Gabriel Gonzales
DS 7333 - QTW
10/26/2020

```
        random_state=123
    )
%%time
import itertools
import numpy as np
import xgboost as xgb

xgtrain = xgb.DMatrix(X_train, y_train)
xgtest = xgb.DMatrix(X_test, y_test)

gamma =  list(np.arange(20, 10, -2))
max_depth = list(np.arange(2, 12, 3))
eta = list(np.arange(0.01, 1, 0.4))
min_child_weight = list(np.arange(0.01, 1, 0.4))
objective="binary:logistic"


parameters = [{"gamma": param[0], "max_depth": param[1], "eta":
param[2],"min_child_weight":param[3], "objective": objective}
              for param in itertools.product(gamma, max_depth, eta, min_child_weight)]

print("Total parameter values to train", len(parameters))

XGBoost Model
def trainXGBoost(param):
    stopping = 3
    boost_rounds = 30

    model = xgb.cv(
        param,
        xgtrain,
        nfold=3,
        metrics=(["logloss",'auc']),
        stratified=True,
        seed=123,
        num_boost_round=boost_rounds,
        early_stopping_rounds=stopping
    )
#     mean_logloss = model['test-logloss-mean'].min()
#     boost_rounds = model['test-logloss-mean'].argmin()
#     mean_auc =  model['test-auc-mean'].max()
#     auc_boost_round = model['test-auc-mean'].argmax()
```

Case Study 8 - XGBoost, Random Forest and SVM

Gabriel Gonzales
DS 7333 - QTW
10/26/2020

```
    return pd.concat([pd.DataFrame([param]), model.tail(1).reset_index()], axis=1)

#     return {
#         'gamma':param["gamma"],
#         'max_depth':param["max_depth"],
#         "eta": param["eta"],
#         "min_child_weight":param["min_child_weight"],
#         "acc":mean_auc,
#         "logloss":mean_logloss,
#         "auc_boost_round": auc_boost_round,
#         "boost_rounds": boost_rounds
#     }
%%time
from tqdm import tqdm
result = list(map(trainXGBoost, tqdm(parameters)))

xgboost_results = pd.concat(result, axis=0).rename(columns={"index":"rounds"})
xgboost_results["rounds"] = xgboost_results.rounds + 1
xgboost_results.to_csv("C:\\Users\\Gabri\XGBOOST_Tuning.csv", index=False)

import seaborn as sns
import matplotlib.pyplot as plt

def heatmapplot(result,param1,param2, metric):
    min_scores = pd.DataFrame(result).groupby([param1, param2])[metric].min().unstack()
    sns.heatmap(min_scores, annot=True, fmt='.4g')

    plt.title('Min score for {} vs {}'.format(param1,param2), fontsize = 15) # title with fontsize 20
    plt.show()
heatmapplot(xgboost_results,'eta','max_depth', 'test-logloss-mean')
heatmapplot(xgboost_results,'eta','gamma', 'test-logloss-mean')
heatmapplot(xgboost_results,'eta','min_child_weight', 'test-logloss-mean')

plt.title('Figure 4: Distribution of rounds by eta', y=-0.25)

ax = sns.boxplot(x="eta", y="rounds", data=xgboost_results)

Random Forest
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import roc_curve
```

Case Study 8 - XGBoost, Random Forest and SVM

Gabriel Gonzales
DS 7333 - QTW
10/26/2020

```
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
import sklearn.feature_selection as fs
from sklearn.model_selection import cross_val_score
import timeit
from sklearn import metrics #Import scikit-learn metrics module for accuracy calculation
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score
from sklearn.metrics import make_scorer
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV

param_grid = {
    "max_features" : ["auto","log2",0.20, 0.30],
    "n_estimators" : [10,50,100],
    "min_samples_leaf" : [25, 50, 100]
}

rfc=RandomForestClassifier(random_state=42)
CV_rfc = GridSearchCV(estimator=rfc, param_grid=param_grid, cv=3,n_jobs = -2, verbose=1)
CV_rfc_mod = CV_rfc.fit(X_train, y_train)

pkl_filename = "CV_Random_Forest.pkl"
with open(pkl_filename, 'wb') as file:
    pickle.dump(CV_rfc, file)

# Load from file
with open("C:\\Users\\Gabri\CV_Random_Forest.pkl", 'rb') as file:
    CV_rf = pickle.load(file)
rf_gridsearch = pd.DataFrame(CV_rf.cv_results_)
rf_columns = [
    "param_max_features",
    "param_n_estimators",
    "param_min_samples_leaf",
    "mean_fit_time",
    "mean_test_score",
    "rank_test_score"
]
```

Case Study 8 - XGBoost, Random Forest and SVM

Gabriel Gonzales
DS 7333 - QTW
10/26/2020

```
import seaborn as sns
import matplotlib.pyplot as plt
plt.title('Figure 5: Distribution of mean_test_score by min_samples_leaf', y=-0.25)
ax = sns.boxplot(x="param_min_samples_leaf", y="mean_test_score", data=rf_gridsearch)

sns.set(style="whitegrid")

plt.title('Figure 6: Distribution of mean_fit_time by max_features', y=-0.25)
ax = sns.boxplot(x="param_max_features", y="mean_fit_time", data=rf_gridsearch)

# Load from file
import pandas as pd
with open("C:\\Users\\Gabri\\CV_Random_Forest_more_parameters.pkl.", 'rb') as file:
    CV_rf2 = pickle.load(file)
rf_columns = [
    "param_max_features",
    "param_n_estimators",
    "param_min_samples_leaf",
    "param_criterion",
    "mean_fit_time",
    "mean_test_accuracy",
    "rank_test_accuracy"
]

rf_gridsearch_2 = pd.DataFrame(CV_rf2.cv_results_)
rf_gridsearch_2[rf_columns].sort_values(['rank_test_accuracy']).head(10)

import plotly.graph_objs as go
import plotly as py

def plotGiniEntropy():
    py.offline.init_notebook_mode(connected=True)
    graph_gini = rf_gridsearch_2[rf_gridsearch_2.param_criterion == 'gini']
    graph_gini =
graph_gini.groupby(['param_max_depth'],sort='False')['mean_test_accuracy'].max()
    graph_gini = graph_gini.to_frame().reset_index()

    graph_entropy = rf_gridsearch_2[rf_gridsearch_2.param_criterion == 'entropy']
    graph_entropy =
graph_entropy.groupby(['param_max_depth'],sort='False')['mean_test_accuracy'].max()
    graph_entropy = graph_entropy.to_frame().reset_index()
```

Case Study 8 - XGBoost, Random Forest and SVM

Gabriel Gonzales
DS 7333 - QTW
10/26/2020

```
# Create traces
trace0 = go.Scatter(
    x = graph_gini.param_max_depth,
    y = graph_gini.mean_test_accuracy,
    mode = 'lines+markers',
    name = 'gini'
)
trace1 = go.Scatter(
    x = graph_entropy.param_max_depth,
    y = graph_entropy.mean_test_accuracy,
    mode = 'lines+markers',
    name = 'entropy'
)

data = [trace0, trace1]

layout = go.Layout(

    title=go.layout.Title(
        text= 'Gini vs Entropy Accuracy over Max Depth',
        xref='container',
#         xanchor = 'center',
        y=.01
    ),
    xaxis=go.layout.XAxis(
        title=go.layout.xaxis.Title(
            text='Max Depth',
            font=dict(
#                 family='Courier New, monospace',
                size=15,
                color='#7f7f7f'
            )
        )
    ),
    yaxis=go.layout.YAxis(
        title=go.layout.yaxis.Title(
            text='Accuracy',
            font=dict(
#                 family='Courier New, monospace',
                size=15,
                color='#7f7f7f'
            )
```

Case Study 8 - XGBoost, Random Forest and SVM

Gabriel Gonzales
DS 7333 - QTW
10/26/2020

```
        )
      )
   )

   fig = go.Figure(data=data, layout=layout)

   fig.show()

   # fig = go.Figure(data=data, layout=layout)

   #py.iplot(fig, filename='line-mode')

# plotGiniEntropy()
plotGiniEntropy()
```

Support Vector Machine Model

```
import plotly.graph_objs as go
import plotly as py

def plotHingeSquare():
   py.offline.init_notebook_mode(connected=True)
   fig = go.Figure()
   graph_hindge = svc_gridsearch[svc_gridsearch.param_loss == 'hinge']
   graph_hindge =
graph_hindge.groupby(['param_max_iter'],sort='False')['mean_test_score'].max()
   graph_hindge = graph_hindge.to_frame().reset_index()

   graph_sqhinge = svc_gridsearch[svc_gridsearch.param_loss == 'squared_hinge']
   graph_sqhinge =
graph_sqhinge.groupby(['param_max_iter'],sort='False')['mean_test_score'].max()
   graph_sqhinge = graph_sqhinge.to_frame().reset_index()

   # Create traces
   trace0 = go.Scatter(
      x = graph_hindge.param_max_iter,
      y = graph_hindge.mean_test_score,
      mode = 'lines+markers',
      name = 'hindge'
   )
   trace1 = go.Scatter(
      x = graph_sqhinge.param_max_iter,
      y = graph_sqhinge.mean_test_score,
```

Case Study 8 - XGBoost, Random Forest and SVM

Gabriel Gonzales
DS 7333 - QTW
10/26/2020

```
    mode = 'lines+markers',
    name = 'square-hinge'
  )

  data = [trace0, trace1]

  layout = go.Layout(

    title=go.layout.Title(
      text= 'Figure 11: Loss: Hinge vs Square Hinge over number or iterations',
      xref='container',
#       xanchor = 'center'
      y=.01
    ),
    xaxis=go.layout.XAxis(
      title=go.layout.xaxis.Title(
        text='Iterations',
        font=dict(
#             family='Courier New, monospace',
          size=15,
          color='#7f7f7f'
        )
      )
    ),
    yaxis=go.layout.YAxis(
      title=go.layout.yaxis.Title(
        text='Test Score',
        font=dict(
#             family='Courier New, monospace',
          size=15,
          color='#7f7f7f'
        )
      )
    )
  )

  fig = go.Figure(data=data, layout=layout)

  fig.show()
### Scale the Dataset
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
```

Case Study 8 - XGBoost, Random Forest and SVM

Gabriel Gonzales
DS 7333 - QTW
10/26/2020

```
from sklearn.model_selection import GridSearchCV
from sklearn.svm import LinearSVC
import sklearn.feature_selection as fs
from sklearn.model_selection import cross_val_score
import timeit
import pickle
from sklearn import metrics #Import scikit-learn metrics module for accuracy calculation
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

df_enc_scaled = scaler.fit_transform(case_8_reduced_feats.values)

X_train, X_test, y_train, y_test = train_test_split(df_enc_scaled, y,
       #stratify=labels,
       test_size=0.30,
       random_state=123)

# Gridsearch to determine the value of C
param_grid = {'C': [0.001,0.01,0.1],
            'loss': ['hinge', 'squared_hinge'],
            'penalty' : ['l2'],
            'dual' : [True,False],
            'tol': [0.00001,0.0001,0.001,0.01,0.1,1], #0.0001 is the Default
            'max_iter': [1500,2000,3000,4000,5000,6000],
           }

SVC_Linear = LinearSVC(random_state=42)
CV_svc = GridSearchCV(estimator = SVC_Linear, param_grid=param_grid, cv= 3, n_jobs = -
1,verbose=1)
CV_svc_mod = CV_rfc.fit(X_train, y_train)

pkl_filename = "CV_SVM_Linear.pkl"
with open(pkl_filename, 'wb') as file:
   pickle.dump(CV_svc_mod, file)

# Load from file
import pandas as pd
import pickle
from sklearn.model_selection import GridSearchCV
from sklearn.svm import LinearSVC
```

Case Study 8 - XGBoost, Random Forest and SVM

Gabriel Gonzales
DS 7333 - QTW
10/26/2020

```
with open("C:\\Users\\Gabri\\CV_SVM_Linear.pkl", 'rb') as file:
    CV_svc = pickle.load(file)
Final Models
import xgboost as xgb

stopping = 3
boost_rounds = 30

xgb_y_hat = np.zeros(len(y))
xgb_y_hat_score = np.zeros(len(y))

for train, test in tqdm(kfold_cv.split(X_scaled,y), total=5):
    xgtrain = xgb.DMatrix(X[train], y[train])
    xgtest = xgb.DMatrix(X[test], y[test])

    bst = xgb.train(xgb_best_param, xgtrain)
    xgb_y_hat[test] = (bst.predict(xgtest) > .5)*1
    xgb_y_hat_score[test] = bst.predict(xgtest)

with open("C:\\Users\\Gabri\\xgb_y_hat.pkl","wb") as f:
    pickle.dump((xgb_y_hat, xgb_y_hat_score), f)

from sklearn.ensemble import RandomForestClassifier

rf_y_hat = np.zeros(len(y))
rf_y_hat_score = np.zeros(len(y))

for train, test in tqdm(kfold_cv.split(X,y), total=5):
    rf_clf = RandomForestClassifier(
        max_features=rf_clf_param["max_features"],
        min_samples_leaf=rf_clf_param["min_samples_leaf"],
        n_estimators=rf_clf_param["n_estimators"],
        random_state=42,
        n_jobs=-2)
    rf_clf.fit(X[train],y[train])
    rf_y_hat[test] = rf_clf.predict(X[test])
    rf_y_hat_score[test] = rf_clf.predict_proba(X[test])[:,1]


with open("C:\\Users\\Gabri\\rf_y_hat.pkl","wb") as f:
    pickle.dump((rf_y_hat, rf_y_hat_score), f)
```

Case Study 8 - XGBoost, Random Forest and SVM

Gabriel Gonzales
DS 7333 - QTW
10/26/2020

```
from sklearn.svm import LinearSVC

svm_y_hat = np.zeros(len(y))

for train, test in tqdm(kfold_cv.split(X_scaled,y), total=5):
    svm_clf = LinearSVC(
        random_state=42,
        #C = svm_clf_param["C"],
        #dual = svm_clf_param["dual"],
        #loss = svm_clf_param["loss"],
        #max_iter = svm_clf_param["max_iter"],
        #penalty = svm_clf_param["penalty"],
        #tol = svm_clf_param["tol"]
    )
    svm_clf.fit(X_scaled[train],y[train])
    svm_y_hat[test] = rf_clf.predict(X[test])



with open("C:\\Users\\Gabri\\svm_y_hat.pkl","wb") as f:
    pickle.dump(svm_y_hat, f)

from sklearn.preprocessing import StandardScaler
from tqdm import tqdm

X = case_8_reduced_feats.values

# SVM Prep
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

rf_clf_param = CV_rf.best_params_
print("Random Forest",rf_clf_param)

svm_clf_param = CV_svc.best_params_
print("SVM",svm_clf_param)
xgb_best_param = xgboost_results.sort_values(by=["test-logloss-mean"],
ascending=True).head(1).reset_index() \
    .T.loc[["gamma","max_depth","eta","min_child_weight","objective"],:].to_dict()[0]
print("XGB", xgb_best_param)
```

Case Study 8 - XGBoost, Random Forest and SVM

Gabriel Gonzales
DS 7333 - QTW
10/26/2020

```
#this analysis was inspired by the cv visualization found at https://scikit-
learn.org/stable/auto_examples/model_selection/plot_cv_indices.html#sphx-glr-auto-examples-
model-selection-plot-cv-indices-py
from sklearn.model_selection import (KFold, ShuffleSplit, StratifiedKFold, StratifiedShuffleSplit)
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.patches import Patch

np.random.seed(42)
cmap_data = plt.cm.Paired
cmap_cv = plt.cm.coolwarm

def cv_plot(ax, cv, X, y):
    plt.figure
    for i, (train_index, test_index) in enumerate(cv.split(X=X, y=y)):
        indices = np.array([np.nan] * len(X))
        indices[test_index] = 1
        indices[train_index] = 0
        ax.scatter(range(len(indices)), [i+1] * len(indices), c=indices, marker='_', lw=10,
cmap=cmap_cv)

    ax.scatter(range(len(indices)), [i+2] * len(indices), c=y, marker='_', lw=10, cmap=cmap_data)
    ytick = list(range(num_cv_iterations)) + ['Class']
    ax.set(yticks=np.arange(num_cv_iterations+1) + 1, yticklabels=ytick, xlabel='Indices',
ylabel="CV iteration")
    ax.set_title('{}'.format(type(cv).__name__), fontsize=15)
    return ax
from sklearn.model_selection import StratifiedKFold

num_cv_iterations = 5
random_st = 42
kfold_cv = StratifiedKFold(
    n_splits=num_cv_iterations,
    shuffle=True,
    random_state = 123
)

fig, ax = plt.subplots()
cv_plot(ax, kfold_cv, X, y)
from sklearn.metrics import accuracy_score, log_loss

rf_y_hat, rf_y_hat_score = pickle.load(open("C:\\Users\\Gabri\\rf_y_hat.pkl","rb"))
```

Case Study 8 - XGBoost, Random Forest and SVM

Gabriel Gonzales
DS 7333 - QTW
10/26/2020

```
svm_y_hat = pickle.load(open("C:\\Users\\Gabri\\svm_y_hat.pkl","rb"))
xgb_y_hat, xgb_y_hat_score = pickle.load(open("C:\\Users\\Gabri\\xgb_y_hat.pkl","rb"))

metric = [
   {
      "metric":"Accuracy",
      "XGBoost": accuracy_score(y, xgb_y_hat),
      "Random Forest": accuracy_score(y, rf_y_hat),
      "Support Vector Machine": accuracy_score(y, svm_y_hat)
   },
   {
      "metric":"Log Loss",
      "XGBoost": log_loss(y, xgb_y_hat_score),
      "Random Forest": log_loss(y, rf_y_hat_score),
      "Support Vector Machine": None
   }
]

pd.DataFrame(metric)

SVM Scaling
%%capture

#### Data Read

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
df_data = pd.read_csv ("C:\\Users\\Gabri\\case_8.csv")
df_data.drop('v22',axis=1,inplace=True)
from tqdm import tqdm
cols = df_data.dtypes
columns = df_data.loc[:,cols == "object"].columns
data = [ pd.get_dummies(df_data[col], prefix=col).copy() for col in tqdm(columns) ]
one_hot_df = pd.concat(data, axis=1)

#Drop old columns

df_data_nc = df_data.drop(columns, axis=1)
df_data_final = pd.concat([df_data_nc, one_hot_df], axis=1)
df_data_encoded = reduce_features(df_data_final, True)
```

Case Study 8 - XGBoost, Random Forest and SVM

Gabriel Gonzales
DS 7333 - QTW
10/26/2020

```
%%capture

from sklearn.model_selection import cross_validate
SVC_Linear_tuned =
LinearSVC(random_state=42,max_iter=7000,C=0.01,loss='squared_hinge',penalty='l2',dual=Fal
se,tol=0.01)

training_time = []
import time
training_sizes = [1000,2000,5000,10000,20000,30000,40000]
for training_size in training_sizes:
    df_data_encoded_train = df_data_encoded.sample(training_size)
    print(len(df_data_encoded_train))

    labels_train = df_data_encoded_train.target
    df_data_encoded_X_train = df_data_encoded_train.drop(["ID","target"],axis = 1)
    df_enc_scaled_train = scaler.fit_transform(df_data_encoded_X_train)
    t0 = time.time()

#     SVC_Linear_tuned.fit(df_enc_scaled_train,labels_train)

    cross_validate(SVC_Linear_tuned,df_enc_scaled_train,labels_train,cv=5)
    t1 = time.time()
    training_time.append((t1-t0))

with open("C:\\Users\\Gabri\\svm_training_time.pkl","wb") as f:
    pickle.dump(training_time, f)

import plotly.express as px

with open("C:\\Users\\Gabri\\svm_training_time.pkl","rb") as f:
    training_time = pickle.load(f)

fig = px.bar(x=training_sizes, y=training_time,range_y = [0,29])

fig.update_layout(xaxis_type='category',
                  xaxis_title="Number of Samples",
                  yaxis_title="Training time in Seconds",
                  title={

                      'y':0.05,
                      'x':0.5,
```

Case Study 8 - XGBoost, Random Forest and SVM

Gabriel Gonzales
DS 7333 - QTW
10/26/2020

```
                'xanchor': 'center',
                'yanchor': 'top'})
fig.show()

from sklearn.linear_model import LinearRegression
from sklearn import metrics

x = np.log(training_sizes).reshape(-1,1)
y = np.log(training_time).reshape(-1,1)

regressor = LinearRegression()
regressor.fit(x, y) #training the algorithm

### Regression Co-Eff
print("Regression (x) coefficient:", regressor.coef_[0][0])
```