Assignment3

November 8, 2025

0.1 Assignment 3

Instructions Perform an analysis of the dataset(s) used in Homework #2 using the SVM algorithm. Compare the results with the results from previous homework. Homework #3

- Read the following articles:
  - https://www.hindawi.com/journals/complexity/2021/5550344/
  - https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8137961/
- Search for academic content (at least 3 articles) that compare the use of decision trees vs SVMs in your current area of expertise.
- Perform an analysis of the dataset used in Homework #2 using the SVM algorithm.
- Compare the results with the results from previous homework.
- Answer questions, such as:
  - Which algorithm is recommended to get more accurate results?
  - Is it better for classification or regression scenarios?
  - Do you agree with the recommendations?
  - Why?

Format

- Essay (minimum 500 word document) Write a short essay explaining your selection of algorithms and how they relate to the data and what you are trying to do.
- Analysis using R or Python (submit code + errors + analysis as notebook or copy/paste to document).Include analysis R (or Python) code.

0.1.1 Two Assigned Articles

Decision Tree Ensembles to Predict Coronavirus Disease 2019 Infection: A Comparative Study (https://www.hindawi.com/journals/complexity/2021/5550344/)

- This paper looked at COvid-19 in patients and attempted to use decision tree modeling on lab results and patient age, with imbalanced outcome considerations, for properly predicting COVID diagnosis in patients. There were 5644 total patients in the original data, with 600 final patients being kept in the study using about 18 different lab biomarker measurements. Of the 600 patents used, 520 were negative and 80 were positive. So about a 13% positivity rate. Data was imbalanced so RUS and SMOTE were used, and the multiple types of Classifier methods were used to obtain results. Overall the best modeling technique was random forest having highest accuracy. with RUSBagging, Balanced Random ForestBalanced, and XGBoost with RUS doing well too.

A novel approach to predict COVID-19 using support vector machine (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8137961/)

- This paper predicts COVID-19 status using a support vector machine modeling on individual's symptoms. The dataset was composed of 200 individual's data with 8 different symptoms": temperature, breathing rate, hypertension, stage heart beat rate, acute respiratory distress syndrome, chest pain, heart disease and cough with sputum. The modeling attempted to categorize individuals into three different classes: not infected, mildly infected, and severely infected. They used a linear SVM model and a 70/30 training/test split. The modeling yielded an overall accuracy of 87 percent, with class results showing the severely infected class with a precision of 0.94, recall of 1.00, and f1 of 0.97. The not infected and mildly infected classes were lower. SVM performed best on this dataset when compared to k-nearest neighbors, naive Bayes, random forest, and AdaBoost.

0.1.2 Three Acadmic Researched Articles

1) A comparative study of forecasting corporate credit ratings using neural networks, support vector machines, and decision trees (https://github.com/jhnboyy/CUNY_SPS_WORK/blob/main/FALL2025/DATA622/Assignmen
%20A%20comparative%20study%20of%20forecasting%20corporate%20credit%20ratings%20using%

- This paper takes a look at corporate credit ratings and examines the models that best forecast them. The data used in this paper covered 52 different financial firms from 1990 through 2018. There were also 28 energy companies and 44 healthcare companies examined for years 2009 through 2018. A total of 16 different metrics were used in order to predict finacial sector ratings. While 20 ratio metrics were used for the energy and healthcare sectors. For the ratings there are 19 different classes from AAA through CC. The data was split for 70/10/20 for train/validate/test sets. When using SVM the authors used one vs. one and one vs. all for the categorization. Bagged Decision Tree performance the best for credit predictions for the financial and healthcare firms examined, while the energy sector was best predicted by Random Forest modeling. The SVM models performed worse.

2)Comparing Support Vector Machines and Decision Trees for Text Classification (https://github.com/jhnboyy/CUNY_SPS_WORK/blob/main/FALL2025/DATA622/Assignmen
%20Comparing%20Support%20Vector%20Machines%20and%20Decision%20Trees%20for%20Text%

- This article compares SVM and decision tree modeling methodologies. Specifically, for text classification using the 20 Newsgroups dataset, with is composed of ~20,000 newsgroup documents, partitioned across 20 different newsgroups. The spllit the data into 80/20 for training/testing. For both models the team vectorizes the word data sets for numeric values suing TfidfVectorizer. The models are then trained. The SVM model achieves an accuracy of about ~91 percent on the test set with strong precision recall and f1. The decision tree reaches ~61 percent accuracy. The article states that in this setting SVM outperforms the decision tree across accuracy precision recall and f1, and explains that SVMs are well suited to high-dimensional text features whereas decision trees are simpler and more interpretable.

-

- This article takes a look at SVM modeling and decision tree modeling in order to predict credit defaults using the UCI Credit Card Fraud dataset, which contains data points on customer's age, gender, education, and loan amount on the default rate. In the data roughly 75 percent made on time payments, while one quarter were late on payments and defaulted. About 3,000 samples were used, however there is no mention of the training test split in the article. The modeling techniques yielded a RMSE of 0.44 with the SVM model and one of 0.42 with the decision tree model. Essentially, the decision tree model performed better with this dataset, having lower error rates.

0.1.3 Performing SVM on Assignment 2 Data

[12]:
```
import pandas as pd
from pandas.plotting import scatter_matrix
import numpy as np
import json
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split, StratifiedKFold,␣
 ↪GridSearchCV, ParameterGrid
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.svm import LinearSVC, SVC
from sklearn.calibration import CalibratedClassifierCV
from sklearn.metrics import (average_precision_score, roc_auc_score,
                             classification_report, confusion_matrix,
                             PrecisionRecallDisplay)
from sklearn.preprocessing import StandardScaler
```

Assignment 2 Replication for the SVM training and Analysis

[2]:
```
### Pulling in the data from Assignment 1
df = pd.read_csv("../Assignment2/raw_df_dropped.csv")
```

[3]:
```
print(df.columns) # No duration because dropped in assignment one. (Data␣
 ↪Leakage)

## Encoding the y var for (1/0) | (yes/no)
df['y'] = df['y'].map({'no':0, 'yes':1})

## Parsing the 'y' from the x vars
X = df.drop(columns=['y'])
y = df['y']

## Splitting the test and train sets for future experiments and modeling
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, stratify=y, random_state=42
)
```

```
# Confirming no nulls; no additional imputation needed because of Assignment 1␣
↪cleaning etc.
# print(X_train.info())
# print(y_train.info())
# print(X_test.info())
# print(y_test.info())
```

```
Index(['age', 'job', 'marital', 'education', 'default', 'balance', 'housing',
       'loan', 'contact', 'day_of_week', 'month', 'campaign', 'previous',
       'poutcome', 'y', 'previously_contacted'],
      dtype='object')
```

[4]:
```
# Getting lists of the x vars and their types For encoding pre-modeling
num_cols = X_train.select_dtypes(include=['number']).columns.tolist()
print(num_cols)
cat_cols = [c for c in X_train.columns if c not in num_cols]
print(cat_cols)
```

```
['age', 'balance', 'day_of_week', 'campaign', 'previous']
['job', 'marital', 'education', 'default', 'housing', 'loan', 'contact',
'month', 'poutcome', 'previously_contacted']
```

[5]:
```
X_train_encoded = pd.get_dummies(X_train, columns=cat_cols, drop_first=False)
X_test_encoded  = pd.get_dummies(X_test,  columns=cat_cols, drop_first=False)
```

[6]:
```
print([i for i in X_train_encoded.columns if i not in X_test_encoded.columns])
print([i for i in X_test_encoded.columns if i not in X_train_encoded.columns])
## Columns are identical, no missing categories in each df so no need for␣
↪alignment
```

```
[]
[]
```

End Assignment Two Replica Code. Beginning of Assignment 3 Changes for SVM modeling

[7]:
```
## Scaling the numeric data
scaler = StandardScaler()
X_train_scaled = X_train_encoded.copy()
X_test_scaled  = X_test_encoded.copy()
X_train_scaled[num_cols] = scaler.fit_transform(X_train_encoded[num_cols])
X_test_scaled[num_cols]  = scaler.transform(X_test_encoded[num_cols])

##  looking  at scaled vals sample
X_train_scaled[num_cols].describe().T.head()
```

[7]:
```
               count          mean       std       min       25%       50%  \
age          35536.0 -2.975258e-16  1.000014 -2.149123 -0.738651 -0.174462
```

```
balance      35536.0  1.829544e-17  1.000014 -2.664600 -0.417440 -0.295665
day_of_week  35536.0 -4.618848e-17  1.000014 -1.776109 -0.935933  0.024268
campaign     35536.0  7.058239e-17  1.000014 -0.571367 -0.571367 -0.244724
previous     35536.0 -3.739067e-17  1.000014 -0.245294 -0.245294 -0.245294

                  75%         max
age          0.671821    5.091300
balance      0.018353   32.722977
day_of_week  0.624393    1.824645
campaign     0.081918   18.047243
previous    -0.245294  113.211503
```

[8]:
```
## Prepping Result Holding Vals
THRESH = 0.5
running_results = {}
rows = []
```

[9]:
```
#Setting up the first linar SVM model
lin_base = LinearSVC(class_weight="balanced", random_state=42)
lin_cal  = CalibratedClassifierCV(lin_base, method="sigmoid", cv=5)

#Fitting the model
lin_cal.fit(X_train_scaled, y_train)

## Predicting values
lin_probs = lin_cal.predict_proba(X_test_scaled)[:, 1]

## adding results to the running dict.
lin_y_pred = (lin_probs >= THRESH).astype(int)

# what C did you use for LinearSVC? (default is 1.0 if you didn't set it)
lin_C = getattr(lin_base, "C", 1.0)
lin_params = {"C": lin_C,   "gamma": "N/A"}

running_results["svm_linear_cal"] = {"prob": lin_probs, "y_pred": lin_y_pred,␣
 ↪"params": lin_params}
```

[14]:
```
## RBF SVM : curved boiundaries, imbalanced y values
rbf = SVC(kernel="rbf", class_weight="balanced", probability=True,␣
 ↪random_state=42)

## Trying different settings for liencncy and finding neighbors
param_grid = {
    "C":     [0.1, 0.25, 0.75, 1, 2, 3, 5, 10],
    "gamma": ["scale", 0.1, 0.06, 0.04, 0.03, 0.02, 0.01]
}
```

```
for params in ParameterGrid(param_grid):
    C = params["C"]; gamma = params["gamma"]
    key = f"svm_rbf_C{C}_g{gamma}"

    print(key)
    # 1) fit this combo
    clf = SVC(kernel="rbf", class_weight="balanced", probability=True,
              random_state=42, C=C, gamma=gamma)
    clf.fit(X_train_scaled, y_train)

    # 2) test predictions
    probs = clf.predict_proba(X_test_scaled)[:, 1]
    y_pred = (probs >= THRESH).astype(int)

    # 3) store minimal artifacts (probabilities + hard preds) -> for your later␣
 ↪aggregation
    running_results[key] = {"prob": probs, "y_pred": y_pred, "params": params}
```

```
svm_rbf_C0.1_gscale
svm_rbf_C0.1_g0.1
svm_rbf_C0.1_g0.06
svm_rbf_C0.1_g0.04
svm_rbf_C0.1_g0.03
svm_rbf_C0.1_g0.02
svm_rbf_C0.1_g0.01
svm_rbf_C0.25_gscale
svm_rbf_C0.25_g0.1
svm_rbf_C0.25_g0.06
svm_rbf_C0.25_g0.04
svm_rbf_C0.25_g0.03
svm_rbf_C0.25_g0.02
svm_rbf_C0.25_g0.01
svm_rbf_C0.75_gscale
svm_rbf_C0.75_g0.1
svm_rbf_C0.75_g0.06
svm_rbf_C0.75_g0.04
svm_rbf_C0.75_g0.03
svm_rbf_C0.75_g0.02
svm_rbf_C0.75_g0.01
svm_rbf_C1_gscale
svm_rbf_C1_g0.1
svm_rbf_C1_g0.06
svm_rbf_C1_g0.04
svm_rbf_C1_g0.03
svm_rbf_C1_g0.02
svm_rbf_C1_g0.01
svm_rbf_C2_gscale
svm_rbf_C2_g0.1
```

```
svm_rbf_C2_g0.06
svm_rbf_C2_g0.04
svm_rbf_C2_g0.03
svm_rbf_C2_g0.02
svm_rbf_C2_g0.01
svm_rbf_C3_gscale
svm_rbf_C3_g0.1
svm_rbf_C3_g0.06
svm_rbf_C3_g0.04
svm_rbf_C3_g0.03
svm_rbf_C3_g0.02
svm_rbf_C3_g0.01
svm_rbf_C5_gscale
svm_rbf_C5_g0.1
svm_rbf_C5_g0.06
svm_rbf_C5_g0.04
svm_rbf_C5_g0.03
svm_rbf_C5_g0.02
svm_rbf_C5_g0.01
svm_rbf_C10_gscale
svm_rbf_C10_g0.1
svm_rbf_C10_g0.06
svm_rbf_C10_g0.04
svm_rbf_C10_g0.03
svm_rbf_C10_g0.02
svm_rbf_C10_g0.01
```

[16]:
```
## Adding the Results Metrics to the Rows Lists via Running Results DIct
for key, value in running_results.items():

    print(key)
    model_prob = value["prob"]
    y_pred = value["y_pred"]
    params = value["params"]
    C = params["C"]
    gamma = params["gamma"]

    classfctn_rpt = classification_report( y_test, y_pred,␣
 ↪target_names=['no','yes'], output_dict=True )

    roc_auc = round(roc_auc_score(y_test, model_prob), 4)
    pr_auc  = round(average_precision_score(y_test, model_prob), 4)
    tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()

    rows.append({
        "model": key,
        "roc_auc": roc_auc,
        "pr_auc": pr_auc,
        "precision_pos": classfctn_rpt["yes"]["precision"],
        "recall_pos":    classfctn_rpt["yes"]["recall"],
        "f1_pos":        classfctn_rpt["yes"]["f1-score"],
        "precision_neg": classfctn_rpt["no"]["precision"],
        "recall_neg":    classfctn_rpt["no"]["recall"],
        "f1_neg":        classfctn_rpt["no"]["f1-score"],
        "macro_precision":   classfctn_rpt["macro avg"]["precision"],
        "macro_recall":      classfctn_rpt["macro avg"]["recall"],
        "macro_f1":          classfctn_rpt["macro avg"]["f1-score"],
        "weighted_precision": classfctn_rpt["weighted avg"]["precision"],
        "weighted_recall":    classfctn_rpt["weighted avg"]["recall"],
        "weighted_f1":        classfctn_rpt["weighted avg"]["f1-score"],
        "tn": tn, "fp": fp, "fn": fn, "tp": tp,
        "C": C, "gamma": gamma
    })
```

```
svm_linear_cal
svm_rbf_C0.1_gscale
svm_rbf_C0.1_g0.1
svm_rbf_C0.1_g0.06
svm_rbf_C0.1_g0.04
svm_rbf_C0.1_g0.03
svm_rbf_C0.1_g0.02
svm_rbf_C0.1_g0.01
svm_rbf_C0.25_gscale
svm_rbf_C0.25_g0.1
svm_rbf_C0.25_g0.06
svm_rbf_C0.25_g0.04
svm_rbf_C0.25_g0.03
svm_rbf_C0.25_g0.02
svm_rbf_C0.25_g0.01
svm_rbf_C0.75_gscale
svm_rbf_C0.75_g0.1
svm_rbf_C0.75_g0.06
svm_rbf_C0.75_g0.04
svm_rbf_C0.75_g0.03
svm_rbf_C0.75_g0.02
svm_rbf_C0.75_g0.01
svm_rbf_C1_gscale
svm_rbf_C1_g0.1
svm_rbf_C1_g0.06
svm_rbf_C1_g0.04
svm_rbf_C1_g0.03
svm_rbf_C1_g0.02
svm_rbf_C1_g0.01
svm_rbf_C2_gscale
svm_rbf_C2_g0.1
```

```
svm_rbf_C2_g0.06
svm_rbf_C2_g0.04
svm_rbf_C2_g0.03
svm_rbf_C2_g0.02
svm_rbf_C2_g0.01
svm_rbf_C3_gscale
svm_rbf_C3_g0.1
svm_rbf_C3_g0.06
svm_rbf_C3_g0.04
svm_rbf_C3_g0.03
svm_rbf_C3_g0.02
svm_rbf_C3_g0.01
svm_rbf_C5_gscale
svm_rbf_C5_g0.1
svm_rbf_C5_g0.06
svm_rbf_C5_g0.04
svm_rbf_C5_g0.03
svm_rbf_C5_g0.02
svm_rbf_C5_g0.01
svm_rbf_C10_gscale
svm_rbf_C10_g0.1
svm_rbf_C10_g0.06
svm_rbf_C10_g0.04
svm_rbf_C10_g0.03
svm_rbf_C10_g0.02
svm_rbf_C10_g0.01
```

[17]:
```python
# All SVM results
svm_results =  pd.DataFrame(rows).round(4)
svm_results['Assignment']="Assignment 3"
```

[25]:
```python
svm_results.shape
```

[25]:
```
(57, 22)
```

[18]:
```python
svm_results.to_csv("svm_results.csv",index=False)
```

[22]:
```python
svm_results.sort_values(by=["roc_auc"], ascending = False).head(n=10)
```

[22]:
```
                  model  roc_auc  pr_auc  precision_pos  recall_pos  f1_pos  \
31      svm_rbf_C2_g0.06   0.7956  0.4174         0.6033      0.1751  0.2714
39      svm_rbf_C3_g0.04   0.7952  0.4238         0.6031      0.1836  0.2815
22     svm_rbf_C1_gscale   0.7950  0.4219         0.6062      0.1874  0.2863
54     svm_rbf_C10_g0.03   0.7948  0.4118         0.6132      0.1675  0.2631
16    svm_rbf_C0.75_g0.1   0.7947  0.4185         0.6051      0.1808  0.2784
15  svm_rbf_C0.75_gscale   0.7947  0.4258         0.6047      0.1951  0.2950
38      svm_rbf_C3_g0.06   0.7947  0.4086         0.5889      0.1513  0.2407
33      svm_rbf_C2_g0.03   0.7946  0.4305         0.6204      0.2084  0.3120
```

```
46      svm_rbf_C5_g0.04   0.7946  0.4130         0.6040      0.1713  0.2669
40      svm_rbf_C3_g0.03   0.7946  0.4283         0.6121      0.1922  0.2925

    precision_neg  recall_neg  f1_neg  macro_precision  …  \
31         0.8989      0.9846  0.9398           0.7511  …
39         0.8998      0.9838  0.9399           0.7515  …
22         0.9002      0.9837  0.9401           0.7532  …
54         0.8982      0.9858  0.9400           0.7557  …
16         0.8995      0.9842  0.9400           0.7523  …
15         0.9010      0.9829  0.9402           0.7529  …
38         0.8964      0.9858  0.9390           0.7427  …
33         0.9025      0.9829  0.9410           0.7614  …
46         0.8986      0.9849  0.9398           0.7513  …
40         0.9007      0.9837  0.9404           0.7564  …

    weighted_precision  weighted_recall  weighted_f1    tn   fp   fn   tp  \
31              0.8640           0.8888       0.8607  7712  121  867  184
39              0.8647           0.8891       0.8620  7706  127  858  193
22              0.8654           0.8895       0.8628  7705  128  854  197
54              0.8645           0.8890       0.8599  7722  111  875  176
16              0.8647           0.8891       0.8617  7709  124  861  190
15              0.8659           0.8897       0.8638  7699  134  846  205
38              0.8601           0.8871       0.8564  7722  111  892  159
33              0.8691           0.8913       0.8666  7699  134  832  219
46              0.8637           0.8887       0.8602  7715  118  871  180
40              0.8666           0.8900       0.8637  7705  128  849  202

        C  gamma    Assignment
31   2.00   0.06  Assignment 3
39   3.00   0.04  Assignment 3
22   1.00  scale  Assignment 3
54  10.00   0.03  Assignment 3
16   0.75    0.1  Assignment 3
15   0.75  scale  Assignment 3
38   3.00   0.06  Assignment 3
33   2.00   0.03  Assignment 3
46   5.00   0.04  Assignment 3
40   3.00   0.03  Assignment 3

[10 rows x 22 columns]
```

[24]:
```
svm_results.sort_values(by=["pr_auc"], ascending = False).head(n=10)
```

[24]:
```
                    model  roc_auc  pr_auc  precision_pos  recall_pos  f1_pos  \
27      svm_rbf_C1_g0.02   0.7920  0.4326         0.5974      0.2160  0.3173
42      svm_rbf_C3_g0.01   0.7917  0.4324         0.5838      0.2055  0.3040
19   svm_rbf_C0.75_g0.03   0.7925  0.4324         0.5954      0.2226  0.3241
```

```
18  svm_rbf_C0.75_g0.04   0.7936  0.4321         0.6052      0.2217  0.3245
20  svm_rbf_C0.75_g0.02   0.7911  0.4320         0.5910      0.2131  0.3133
49     svm_rbf_C5_g0.01   0.7927  0.4319         0.6053      0.2160  0.3184
26     svm_rbf_C1_g0.03   0.7932  0.4315         0.6061      0.2255  0.3287
35     svm_rbf_C2_g0.01   0.7904  0.4311         0.5881      0.2065  0.3056
25     svm_rbf_C1_g0.04   0.7940  0.4308         0.6006      0.2074  0.3083
33     svm_rbf_C2_g0.03   0.7946  0.4305         0.6204      0.2084  0.3120

    precision_neg  recall_neg  f1_neg  macro_precision  …  \
27         0.9031      0.9805  0.9402           0.7502  …
42         0.9019      0.9803  0.9395           0.7429  …
19         0.9038      0.9797  0.9402           0.7496  …
18         0.9038      0.9806  0.9406           0.7545  …
20         0.9028      0.9802  0.9399           0.7469  …
49         0.9032      0.9811  0.9405           0.7542  …
26         0.9042      0.9803  0.9407           0.7551  …
35         0.9021      0.9806  0.9397           0.7451  …
25         0.9022      0.9815  0.9402           0.7514  …
33         0.9025      0.9829  0.9410           0.7614  …

    weighted_precision  weighted_recall  weighted_f1    tn   fp   fn   tp  \
27              0.8669           0.8900       0.8665  7680  153  824  227
42              0.8643           0.8887       0.8643  7679  154  835  216
19              0.8673           0.8901       0.8673  7674  159  817  234
18              0.8684           0.8908       0.8677  7681  152  818  233
20              0.8659           0.8895       0.8658  7678  155  827  224
49              0.8679           0.8906       0.8669  7685  148  824  227
26              0.8689           0.8910       0.8683  7679  154  814  237
35              0.8649           0.8890       0.8647  7681  152  834  217
25              0.8666           0.8899       0.8654  7688  145  833  218
33              0.8691           0.8913       0.8666  7699  134  832  219

       C  gamma    Assignment
27  1.00   0.02  Assignment 3
42  3.00   0.01  Assignment 3
19  0.75   0.03  Assignment 3
18  0.75   0.04  Assignment 3
20  0.75   0.02  Assignment 3
49  5.00   0.01  Assignment 3
26  1.00   0.03  Assignment 3
35  2.00   0.01  Assignment 3
25  1.00   0.04  Assignment 3
33  2.00   0.03  Assignment 3

[10 rows x 22 columns]
```

Notes on SVm Model Results

- After running 57 RBF kernel SVM models with different parameters for gamma and C, along with one linar SBF model, the model that performed the best was the "svm_rbf_C1_g0.02" model. SPecifically, when looking at ROC_AUC the RBF SVM model with C at 2 and gamma at 0.06 obtained a score of 0.7956 (79.56%). The runner up was "svm_rbf_C3_g0.04", the RBF SVM model with C at 3 and gamma at 0.04, obtained a ROC_AUC score of 79.52%.
- WHen looking at PR-AUC, the top performing model was "svm_rbf_C1_g0.02", which was the RBF SVM model with C at 1 and gamma at 0.02. THis model obtained a 0.4326 pr-auc. The runner up for PR-AUC scores was "svm_rbf_C3_g0.01", which was the RBF SVM model with C at 3 and gamma at 0.01.

Looking at Top 3 SVM models

```
[27]: svm_results.sort_values(by=["roc_auc"], ascending = False).head(n=3)
```

```
[27]:               model  roc_auc  pr_auc  precision_pos  recall_pos  f1_pos  \
31   svm_rbf_C2_g0.06   0.7956  0.4174         0.6033      0.1751  0.2714
39   svm_rbf_C3_g0.04   0.7952  0.4238         0.6031      0.1836  0.2815
22  svm_rbf_C1_gscale   0.7950  0.4219         0.6062      0.1874  0.2863

    precision_neg  recall_neg  f1_neg  macro_precision  …  \
31         0.8989      0.9846  0.9398           0.7511  …
39         0.8998      0.9838  0.9399           0.7515  …
22         0.9002      0.9837  0.9401           0.7532  …

    weighted_precision  weighted_recall  weighted_f1    tn   fp   fn   tp  \
31              0.8640           0.8888       0.8607  7712  121  867  184
39              0.8647           0.8891       0.8620  7706  127  858  193
22              0.8654           0.8895       0.8628  7705  128  854  197

      C  gamma    Assignment
31  2.0   0.06  Assignment 3
39  3.0   0.04  Assignment 3
22  1.0  scale  Assignment 3

[3 rows x 22 columns]
```

Pulling in Top Performing models from Assignment 2

```
[15]: assgn2_best = pd.read_csv("../Assignment2/Assignment2_top3_model_results.csv")
      assgn2_best['Assignment']="Assignment 2"
```

```
[15]:           model  roc_auc  pr_auc  precision_pos  recall_pos  f1_pos  \
0  rand_forest3   0.8017  0.4422         0.4509      0.5243  0.4848
1  rand_forest2   0.7958  0.4350         0.4971      0.4139  0.4517
2     adaboost1   0.7851  0.4289         0.6285      0.1931  0.2955

   precision_neg  recall_neg  f1_neg  macro_precision  macro_recall  macro_f1  \
0         0.9347      0.9143  0.9244           0.6928        0.7193    0.7046
1         0.9231      0.9438  0.9333           0.7101        0.6789    0.6925
```

```
2         0.9009      0.9847  0.9410           0.7647         0.5889    0.6182

   weighted_precision  weighted_recall  weighted_f1    tn   fp   fn   tp
0              0.8775           0.8682       0.8724  7162  671  500  551
1              0.8727           0.8811       0.8764  7393  440  616  435
2              0.8687           0.8910       0.8646  7713  120  848  203
```

Notes on Comparing Assignment 2 results to the top 3 SVM results.

- When comparing the results from Assignment 2, which were the bext decision tree models on the same data, the random forest model number 3 was still the best performing ROC-AUC scors wt 0.8017 versus the best SVM model which is at 0.7956. The scores are very similar, but the Random Forest model is better.
- The second-highest scoring model from the decision tree cohort is still just slightly higher than the best performing Svm model when looking at the ROC_AUC score. THe random forest 2 model has a score of 0.7958, while the SVM model has 0.7956.

0.1.4 Qestions to Answer

Which algorithm is recommended to get more accurate results?

- Overall, the best performing model accross Assignment 2 and Assignment 3, was the Random Forest #3 model. WIth the highest scores for roc-auc. So the recommended model is rand_forest_3 from assignment 2, not any SVM model from Assignment 3.

Is it better for classification or regression scenarios?

- This data exercise was a classification scenario, the decision tree model did perform better, just slightly, as the top SVM model was a close runner up. It seems SVm is well equiped to handle categorization scenarions, while decision trees are pretty well equipped to handle either or. This secnario had an imbalanced target and also a decent amount of tabular data, which may have impacted its higher results.

Do you agree with the recommendations? Why?

- Yes, i agree that the recommendation based on these results would be to suggest a decision tree, more specifically a random forest modeling methodology, as it obtained better results. However, that being said the SVM methodology did obtain a very close second place score. Additionally, when looking at the papers reviewed for this assignment, nearly all of the papers found that specific deicison tree methodologies out performed the SVM methodologies for perdicitve modeling on the data. In each of the researched papers, with the exception of the text classification scenario where SVM performed better, the decision tree methodologies were best. In conclusion, for thie scenario, where the issues isnt text classification, but rather more tabular data with imbalanced outcomes for reference, decision trees are better thant SVMs.

```
[ ]:
```

```
[ ]:
```

```
[ ]:
```

```
[ ]:
```

```
[ ]:
```

```
[ ]:
```