insurance

February 27, 2024

```
[ ]: from sklearn.impute import SimpleImputer
     import pandas as pd
     from scipy.stats import spearmanr

     #
     df = pd.read_csv(r'/kaggle/input/dataset/data.csv', encoding='gbk')
     #
     imputer = SimpleImputer(strategy='median')  #
     df_imputed = pd.DataFrame(imputer.fit_transform(df), columns=df.columns)
     #
     correlations = {}
     for feature in df_imputed.columns[:-1]:
         if len(df_imputed[feature].unique()) == 1:
             print(f"Skipping constant feature: {feature}")
             continue
         #
         if df_imputed[feature].var() == 0:
             print(f"Skipping feature with zero variance: {feature}")
             continue
         corr, _ = spearmanr(df_imputed[feature], df_imputed[df_imputed.
      ↪columns[-1]], nan_policy='omit')
         correlations[feature] = abs(corr)

     #
     sorted_correlations = sorted(correlations.items(), key=lambda x: x[1],␣
      ↪reverse=True)
     top_15_features = [f for f in sorted_correlations ][:15]
     print("Top 15 features without significant collinearity:")
     for feature, corr in top_15_features:
         print(f"{feature}: {corr}")
     import numpy as np
     #   15
     selected_features = [f[0] for f in top_15_features]
```

```
Skipping constant feature:   YYYY_NN
Skipping constant feature:   _SUM
Top 15 features without significant collinearity:
    _SUM: 0.2983580665961224
```

```
        _SUM: 0.29680487519963766
    _MAX: 0.17305821984274666
    _MAX: 0.15909622257824305
    _MAX: 0.15571888197269979
    _MAX: 0.13386501179512825
    _AVG: 0.1320621397943678
    _AVG: 0.12934033546139856
       : 0.12868576122647152
    _AVG: 0.1282846286397189
     _SUM: 0.11497372613859341
      _SUM: 0.11485987398900502
     _SUM: 0.11416324888696505
     _SUM: 0.1138687327779898
    _SUM: 0.11292278007337614
```

[ ]:
```
#   DataFrame         float32
X = df_imputed.loc[:, selected_features].values.astype(np.float32)

#       float32
y = df.iloc[:, -1].values.reshape(-1, 1).astype(np.float32)
#   ndarray
print("Shape of X:", X.shape)
print("Shape of y:", y.shape)
```

```
Shape of X: (16000, 15)
Shape of y: (16000, 1)
```

[ ]:
```
import numpy as np

min_vals = np.min(X, axis=0)
max_vals = np.max(X, axis=0)
denominator = max_vals - min_vals
denominator[denominator == 0] = 1  #    0   1   0
#    Min-Max
normalized_X = (X - min_vals) / denominator
from sklearn.model_selection import train_test_split
X_train, X_val, y_train, y_val = train_test_split(normalized_X, y, test_size=0.
 ↪3)
from sklearn.ensemble import GradientBoostingClassifier
from imblearn.over_sampling import SMOTE
from sklearn.metrics import accuracy_score, recall_score
smote = SMOTE(k_neighbors=5, random_state=42,sampling_strategy=0.3)
trainX, trainY = smote.fit_resample(X_train, y_train)
```

[ ]:
```
from sklearn.svm import SVC
svmModel = SVC(kernel='rbf',gamma='auto',class_weight={0: 1, 1: 6})
svmModel.fit(trainX, trainY )
svmLabels = svmModel.predict(X_val)
```

```
accuracy = accuracy_score(y_val, svmLabels)
print('SVM   :', accuracy)
recall = recall_score(y_val, svmLabels, pos_label=1)
print('  :', recall)
```

```
SVM   : 0.7783333333333333
  : 0.6536964980544747
```

[ ]:
```
from joblib import dump
#
dump(svmModel, 'svmModel.joblib')
```

[ ]:
```
['svmModel.joblib']
```

[ ]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import accuracy_score, recall_score

#
t = DecisionTreeClassifier(max_leaf_nodes=10, random_state=42)
#  AdaBoost
boostedTreeModel = AdaBoostClassifier(
    estimator=t,
    n_estimators=100,
    algorithm='SAMME',
    random_state=42
)
sample_weight = [1 if y == 0 else 10 for y in trainY]
boostedTreeModel.fit(trainX, trainY, sample_weight=sample_weight)
boostedTreeLabels = boostedTreeModel.predict(X_val)
accuracy = accuracy_score(y_val, boostedTreeLabels)
print('Boosted Tree Accuracy:', accuracy)
recall = recall_score(y_val, boostedTreeLabels, pos_label=1)
print('Recall:', recall)
```

```
Boosted Tree Accuracy: 0.8333333333333334
Recall: 0.7704280155642024
```

[ ]:
```
from joblib import dump
#
dump(boostedTreeModel, 'boostedTreeModel.joblib')
```

[ ]:
```
['boostedTreeModel.joblib']
```

[ ]:
```
from sklearn.linear_model import LogisticRegression
class_weights = {0: 1, 1: 5}
logistic_model = LogisticRegression(class_weight=class_weights,random_state=42)
logistic_model.fit(trainX, trainY)
```

```
logistic_labels = logistic_model.predict(X_val)
accuracy = accuracy_score(y_val, logistic_labels)
print('Logistic Regression Accuracy:', accuracy)
recall = recall_score(y_val, logistic_labels, pos_label=1)
print('Recall:', recall)
```

```
Logistic Regression Accuracy: 0.7175
Recall: 0.7120622568093385
```

[ ]:

```
from joblib import dump
#
dump(logistic_model, 'logistic_model.joblib')
```

[ ]:

```
['logistic_model.joblib']
```

[ ]:

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.metrics import accuracy_score

#        10    1
class_weights = {0: 1, 1: 5}

#
trainX_weighted = []
trainY_weighted = []
for i, x in enumerate(trainX):
    trainX_weighted.append(x)
    trainY_weighted.append(trainY[i])
    if trainY[i] == 1:
        #
        for _ in range(class_weights[1] - 1):
            trainX_weighted.append(x)
            trainY_weighted.append(trainY[i])
lda_model = LinearDiscriminantAnalysis()
lda_model.fit(trainX_weighted, trainY_weighted)
lda_labels = lda_model.predict(X_val)
accuracy = accuracy_score(y_val, lda_labels)
print('Fisher Linear Discriminant Analysis Accuracy:', accuracy)
recall = recall_score(y_val, lda_labels, pos_label=1)
print('Recall:', recall)
```

```
Fisher Linear Discriminant Analysis Accuracy: 0.7041666666666667
Recall: 0.7042801556420234
```

[ ]:

```
from joblib import dump
#
dump(lda_model, 'lda_model.joblib')
```

[ ]:

```
['lda_model.joblib']
```

```
[ ]: from sklearn.ensemble import RandomForestClassifier
     from sklearn.metrics import accuracy_score
     class_weight = {0: 1, 1: 100}
     #
     clf = RandomForestClassifier(n_estimators=100, random_state=42)
     sample_weight = [1 if y == 0 else 100 for y in trainY]
     #
     clf.fit(trainX, trainY,sample_weight=sample_weight)

     #
     predictions = clf.predict(X_val)

     #
     accuracy = accuracy_score(y_val, predictions)
     print("Accuracy:", accuracy)
     #
     recall = recall_score(y_val, predictions, pos_label=1)
     print('  :', recall)

     from joblib import dump
     #
     dump(clf, 'random_forest_model.joblib')
```

```
Accuracy: 0.9502083333333333
  : 0.33852140077821014
```

```
[ ]: ['random_forest_model.joblib']
```

```
[ ]: from sklearn.ensemble import AdaBoostClassifier
     from sklearn.metrics import accuracy_score, recall_score
     from joblib import dump
     class_weights = {0: 1., 1: 5.}  #    1    10
     #  AdaBoost
     boosted_model = AdaBoostClassifier(
         n_estimators=60,
         algorithm='SAMME',
         random_state=42
     )
     #
     boosted_model.fit(trainX, trainY, sample_weight=[class_weights[yi] for yi in␣
      ↪trainY])
     #
     boosted_labels = boosted_model.predict(X_val)
     #
     accuracy = accuracy_score(y_val, boosted_labels)
     recall = recall_score(y_val, boosted_labels, pos_label=1)
     print('Boosted Tree Accuracy:', accuracy)
     print('Recall:', recall)
```

```
#
dump(boosted_model, 'boosted_model.joblib')

# #
# loaded_model = load('boosted_model.joblib')
```

```
Boosted Tree Accuracy: 0.7752083333333334
Recall: 0.7976653696498055
```

```
[ ]: ['boosted_model.joblib']
```

```
[ ]: from keras import Sequential
from keras.layers import Dense,Dropout,BatchNormalization,Activation,Dropout
model=Sequential()
model.add(Dense(512))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.3))
model.add(Dense(256))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.3))
# model.add(Dense(64))
# model.add(BatchNormalization())
# model.add(Activation('relu'))
model.add(Dense(1))
model.add(Activation('sigmoid'))

model.compile(optimizer='adam', loss='binary_crossentropy')
class_weights={0:1,1:5}
history=model.fit(trainX,trainY,
                  epochs=100,
                  shuffle=True,
                  class_weight=class_weights)
```

```
Epoch 1/100
434/434 [==============================] - 3s 4ms/step - loss: 0.9721
Epoch 2/100
434/434 [==============================] - 2s 4ms/step - loss: 0.8707
Epoch 3/100
434/434 [==============================] - 2s 4ms/step - loss: 0.8301
Epoch 4/100
434/434 [==============================] - 2s 4ms/step - loss: 0.8219
Epoch 5/100
434/434 [==============================] - 2s 4ms/step - loss: 0.8127
Epoch 6/100
434/434 [==============================] - 2s 4ms/step - loss: 0.8112
Epoch 7/100
```

```
434/434 [==============================] - 2s 4ms/step - loss: 0.7842
Epoch 8/100
434/434 [==============================] - 2s 4ms/step - loss: 0.7764
Epoch 9/100
434/434 [==============================] - 2s 4ms/step - loss: 0.7711
Epoch 10/100
434/434 [==============================] - 2s 4ms/step - loss: 0.7789
Epoch 11/100
434/434 [==============================] - 2s 4ms/step - loss: 0.7433
Epoch 12/100
434/434 [==============================] - 2s 4ms/step - loss: 0.7427
Epoch 13/100
434/434 [==============================] - 2s 4ms/step - loss: 0.7454
Epoch 14/100
434/434 [==============================] - 2s 4ms/step - loss: 0.7424
Epoch 15/100
434/434 [==============================] - 2s 4ms/step - loss: 0.7280
Epoch 16/100
434/434 [==============================] - 2s 4ms/step - loss: 0.7373
Epoch 17/100
434/434 [==============================] - 2s 4ms/step - loss: 0.7195
Epoch 18/100
434/434 [==============================] - 2s 4ms/step - loss: 0.7070
Epoch 19/100
434/434 [==============================] - 2s 4ms/step - loss: 0.7176
Epoch 20/100
434/434 [==============================] - 2s 4ms/step - loss: 0.7000
Epoch 21/100
434/434 [==============================] - 2s 4ms/step - loss: 0.7029
Epoch 22/100
434/434 [==============================] - 2s 4ms/step - loss: 0.6996
Epoch 23/100
434/434 [==============================] - 2s 4ms/step - loss: 0.7061
Epoch 24/100
434/434 [==============================] - 2s 4ms/step - loss: 0.7000
Epoch 25/100
434/434 [==============================] - 2s 4ms/step - loss: 0.6866
Epoch 26/100
434/434 [==============================] - 2s 4ms/step - loss: 0.6874
Epoch 27/100
434/434 [==============================] - 2s 4ms/step - loss: 0.6857
Epoch 28/100
434/434 [==============================] - 2s 4ms/step - loss: 0.6952
Epoch 29/100
434/434 [==============================] - 2s 4ms/step - loss: 0.6792
Epoch 30/100
434/434 [==============================] - 2s 4ms/step - loss: 0.6807
Epoch 31/100
```

```
434/434 [==============================] - 2s 4ms/step - loss: 0.6752
Epoch 32/100
434/434 [==============================] - 2s 4ms/step - loss: 0.6786
Epoch 33/100
434/434 [==============================] - 2s 4ms/step - loss: 0.6754
Epoch 34/100
434/434 [==============================] - 2s 4ms/step - loss: 0.6657
Epoch 35/100
434/434 [==============================] - 2s 4ms/step - loss: 0.6698
Epoch 36/100
434/434 [==============================] - 2s 4ms/step - loss: 0.6725
Epoch 37/100
434/434 [==============================] - 2s 4ms/step - loss: 0.6744
Epoch 38/100
434/434 [==============================] - 2s 4ms/step - loss: 0.6648
Epoch 39/100
434/434 [==============================] - 2s 5ms/step - loss: 0.6617
Epoch 40/100
434/434 [==============================] - 2s 4ms/step - loss: 0.6611
Epoch 41/100
434/434 [==============================] - 2s 4ms/step - loss: 0.6514
Epoch 42/100
434/434 [==============================] - 2s 4ms/step - loss: 0.6552
Epoch 43/100
434/434 [==============================] - 2s 4ms/step - loss: 0.6513
Epoch 44/100
434/434 [==============================] - 2s 4ms/step - loss: 0.6549
Epoch 45/100
434/434 [==============================] - 2s 4ms/step - loss: 0.6450
Epoch 46/100
434/434 [==============================] - 2s 4ms/step - loss: 0.6408
Epoch 47/100
434/434 [==============================] - 2s 4ms/step - loss: 0.6457
Epoch 48/100
434/434 [==============================] - 2s 4ms/step - loss: 0.6443
Epoch 49/100
434/434 [==============================] - 2s 4ms/step - loss: 0.6398
Epoch 50/100
434/434 [==============================] - 2s 4ms/step - loss: 0.6557
Epoch 51/100
434/434 [==============================] - 2s 4ms/step - loss: 0.6428
Epoch 52/100
434/434 [==============================] - 2s 4ms/step - loss: 0.6340
Epoch 53/100
434/434 [==============================] - 2s 4ms/step - loss: 0.6402
Epoch 54/100
434/434 [==============================] - 2s 4ms/step - loss: 0.6353
Epoch 55/100
```

```
434/434 [==============================] - 2s 4ms/step - loss: 0.6402
Epoch 56/100
434/434 [==============================] - 2s 4ms/step - loss: 0.6399
Epoch 57/100
434/434 [==============================] - 2s 4ms/step - loss: 0.6210
Epoch 58/100
434/434 [==============================] - 2s 4ms/step - loss: 0.6426
Epoch 59/100
434/434 [==============================] - 2s 4ms/step - loss: 0.6178
Epoch 60/100
434/434 [==============================] - 2s 4ms/step - loss: 0.6179
Epoch 61/100
434/434 [==============================] - 2s 4ms/step - loss: 0.6324
Epoch 62/100
434/434 [==============================] - 2s 4ms/step - loss: 0.6169
Epoch 63/100
434/434 [==============================] - 2s 4ms/step - loss: 0.6144
Epoch 64/100
434/434 [==============================] - 2s 4ms/step - loss: 0.6189
Epoch 65/100
434/434 [==============================] - 2s 4ms/step - loss: 0.6273
Epoch 66/100
434/434 [==============================] - 2s 4ms/step - loss: 0.6168
Epoch 67/100
434/434 [==============================] - 2s 4ms/step - loss: 0.6167
Epoch 68/100
434/434 [==============================] - 2s 4ms/step - loss: 0.6114
Epoch 69/100
434/434 [==============================] - 2s 4ms/step - loss: 0.6127
Epoch 70/100
434/434 [==============================] - 2s 4ms/step - loss: 0.6141
Epoch 71/100
434/434 [==============================] - 2s 4ms/step - loss: 0.6185
Epoch 72/100
434/434 [==============================] - 2s 4ms/step - loss: 0.6164
Epoch 73/100
434/434 [==============================] - 2s 4ms/step - loss: 0.6057
Epoch 74/100
434/434 [==============================] - 2s 4ms/step - loss: 0.6145
Epoch 75/100
434/434 [==============================] - 2s 4ms/step - loss: 0.6102
Epoch 76/100
434/434 [==============================] - 2s 4ms/step - loss: 0.6176
Epoch 77/100
434/434 [==============================] - 2s 4ms/step - loss: 0.6012
Epoch 78/100
434/434 [==============================] - 2s 4ms/step - loss: 0.6056
Epoch 79/100
```

```
434/434 [==============================] - 2s 4ms/step - loss: 0.6103
Epoch 80/100
434/434 [==============================] - 2s 4ms/step - loss: 0.6042
Epoch 81/100
434/434 [==============================] - 2s 4ms/step - loss: 0.5940
Epoch 82/100
434/434 [==============================] - 2s 4ms/step - loss: 0.6013
Epoch 83/100
434/434 [==============================] - 2s 4ms/step - loss: 0.5938
Epoch 84/100
434/434 [==============================] - 2s 4ms/step - loss: 0.5963
Epoch 85/100
434/434 [==============================] - 2s 4ms/step - loss: 0.5888
Epoch 86/100
434/434 [==============================] - 2s 4ms/step - loss: 0.6018
Epoch 87/100
434/434 [==============================] - 2s 4ms/step - loss: 0.6025
Epoch 88/100
434/434 [==============================] - 2s 4ms/step - loss: 0.5968
Epoch 89/100
434/434 [==============================] - 2s 4ms/step - loss: 0.5940
Epoch 90/100
434/434 [==============================] - 2s 4ms/step - loss: 0.5771
Epoch 91/100
434/434 [==============================] - 2s 4ms/step - loss: 0.5954
Epoch 92/100
434/434 [==============================] - 2s 4ms/step - loss: 0.6058
Epoch 93/100
434/434 [==============================] - 2s 4ms/step - loss: 0.5818
Epoch 94/100
434/434 [==============================] - 2s 4ms/step - loss: 0.5816
Epoch 95/100
434/434 [==============================] - 2s 4ms/step - loss: 0.5775
Epoch 96/100
434/434 [==============================] - 2s 4ms/step - loss: 0.5856
Epoch 97/100
434/434 [==============================] - 2s 4ms/step - loss: 0.5768
Epoch 98/100
434/434 [==============================] - 2s 4ms/step - loss: 0.6008
Epoch 99/100
434/434 [==============================] - 2s 4ms/step - loss: 0.5792
Epoch 100/100
434/434 [==============================] - 2s 4ms/step - loss: 0.5876
```

[ ]:
```
predictions = model.predict(X_val)
y_pred_classes = (predictions > 0.5).astype(int)
from sklearn.metrics import accuracy_score ,recall_score
```

```
accuracy = accuracy_score(y_val, y_pred_classes)
print(f"Accuracy: {accuracy}")


#         1
recall = recall_score(y_val, y_pred_classes)
print(f"Recall: {recall}")
```

```
150/150 [==============================] - 0s 1ms/step
Accuracy: 0.809375
Recall: 0.8365758754863813
```

[ ]:
```
model.save('model.keras')
```

[ ]:
```
from joblib import load
from keras.models import load_model
model1=load('boostedTreeModel.joblib')
model2=load('lda_model.joblib')
model3=load('boosted_model.joblib')
model4=load('logistic_model.joblib')
model5=load('svmModel.joblib')
model6=load('random_forest_model.joblib')
model7=load_model('model.keras')

#
predictions1 = np.where(model1.predict(X_val) > 0.5, 1, 0)
predictions2 = np.where(model2.predict(X_val) > 0.5, 1, 0)
predictions3 = np.where(model3.predict(X_val) > 0.5, 1, 0)
predictions4 = np.where(model4.predict(X_val) > 0.5, 1, 0)
predictions5 = np.where(model5.predict(X_val) > 0.5, 1, 0)
predictions6 = np.where(model6.predict(X_val) > 0.5, 1, 0)
predictions7 = np.where(model7.predict(X_val) > 0.5, 1, 0)

#
weights = [0.3, 0.1, 0.5, 0.1, 0.2, 0.1,1]  #

#

ans0=weights[0] * predictions1
ans1=weights[1] * predictions2
ans2=weights[2] * predictions3
ans3=weights[3] * predictions4
ans4=weights[4] * predictions5
ans5=weights[5] * predictions6
ans6=(weights[6] * predictions7).reshape(-1,)
```

```
150/150 [==============================] - 0s 1ms/step
```

```
[ ]: print(ans0.shape)
     print(ans1.shape)
     print(ans2.shape)
     print(ans3.shape)
     print(ans4.shape)
     print(ans5.shape)
     print(ans6.shape)
```

```
(4800,)
(4800,)
(4800,)
(4800,)
(4800,)
(4800,)
(4800,)
```

```
[ ]: ans=(ans0+ans1+ans2+ans3+ans4+ans5+ans6)/np.sum(weights)
```

```
[ ]: #         0.5
     binary_predictions = (ans > 0.5).astype(int)

     #
     print("Binary Predictions:", binary_predictions)

     #
     accuracy = accuracy_score(y_val, binary_predictions)
     print("Weighted Classification Accuracy:", accuracy)

     recall=recall_score(y_val, binary_predictions)
     print("Weighted Classification Recall:", recall)
```

```
Binary Predictions: [0 1 1 … 0 0 0]
Weighted Classification Accuracy: 0.8233333333333334
Weighted Classification Recall: 0.8287937743190662
```