Title: Innovative Solutions for imbalanced Data in Diabetes Prediction: A Multifaceted Machine Learning Approach

Machine Learning(DSCI-6003-03)

Team Members:

- **Srikanth Thota and 00887643**
- **Dinesh Reddy Jetta and 00886296**
- **Rishikeshwar Kankurthyshetty and 00854211**
- **Sesha Sai Veerla and 00877627**

Department of Data Science

University of New Haven

21 April 2024

Abstract:

Unbalanced data, when examples of one class greatly exceed those of the other, is a common problem for machine learning-based diabetes prediction. Our method uses several cutting-edge techniques to overcome this. First, to obtain a more equitable representation of the dataset, we use data augmentation techniques to create synthetic samples for the minority class. Second, we use ensemble learning techniques, like bagging, boosting, and stacking, to integrate predictions from several models to identify various patterns in the data and enhance overall performance. To accommodate for different misclassification costs, we also incorporate cost-sensitive learning into our architecture, so that the model prioritizes decreasing errors that have the greatest impact. In addition, we apply feature selection methods to find the most informative predictors for diabetes prediction, improving interpretability and accuracy.

Introduction:

Early diagnosis and successful diabetes management depend on machine learning-based diabetes prediction. But accurate prediction faces difficulties when dealing with imbalanced datasets, in which one class greatly surpasses the other. We address the problem of imbalanced data in diabetes prediction in this research by presenting a multimodal machine learning approach. Our method seeks to improve prediction model efficacy and dependability by using novel approaches such feature selection, cost-sensitive learning, ensemble learning, and data augmentation. We show that our method is effective in enhancing prediction ability and reducing the effects of class imbalance through thorough testing and assessment. This study advances the use of predictive analytics in healthcare by providing new methods for improving the accuracy of diabetes predictions.

Methods:

To accomplish the project's goals, the following instruments and methods will be applied:

- **Data Preprocessing:** The dataset will be cleansed, missing values will be appended, and anomalies will be found and eliminated.
- **Feature selection:** The most important risk factors for diabetes prediction will be found using a feature selection technique.
- **Resolving class imbalance:** The imbalanced nature of the dataset will be addressed using various techniques such as oversampling, under sampling, and Synthetic Minority Over-sampling Technique.
- **Multiple machine learning techniques:** The development of a predictive model will involve the use of several machine learning methods, including support vector machine (SVM), logistic regression, decision trees, random forests, and others.
- **Model evaluation:** A variety of performance indicators, including the confusion matrix, area under the receiver operating characteristic (ROC) curve, and F1 score, will be used to assess the constructed model's accuracy, sensitivity, specificity, precision, and recall.

Results:

In the context of diabetes prediction, imbalanced datasets were shown to be a major barrier that reduces the precision of predictive algorithms. Using techniques like ensembles and Deep learning, this research pioneers a holistic strategy inside machine learning to specifically address the issue of class imbalance. Enhancing projected accuracy is our primary objective, and we want to contribute significantly to the creation of cutting-edge clinical decision support systems. By strengthening the identification of individuals who are at risk of diabetes, we hope to lay the groundwork for more robust and dependable predictive models in clinical settings by increasing our understanding of and use of various machine learning techniques.

```
# Generate descriptive statistics for the DataFrame
data.describe()
```

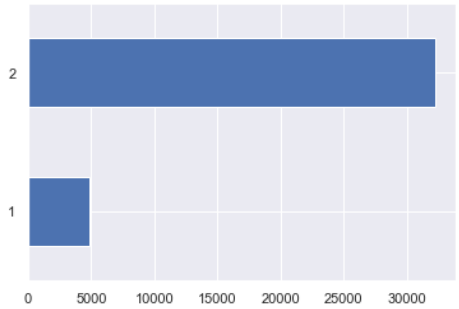
]:

	Gender	Age	Systolic	Diastolic	Weight	Body-Mass-Index	Hemoglobin	Albumin	ALP	AST	...
count	37079.000000	37079.000000	37079.000000	37079.000000	37079.000000	37079.000000	37079.000000	37079.000000	37079.000000	37079.000000	...
mean	1.513282	48.943661	124.090078	69.919253	80.988276	28.824588	14.139073	42.528116	70.789611	25.722511	...
std	0.499830	18.010440	19.254741	13.575804	20.678734	6.608982	1.541599	3.585254	26.073559	19.695625	...
min	1.000000	20.000000	0.000000	0.000000	32.300000	13.180000	5.800000	19.000000	7.000000	7.000000	...
25%	1.000000	33.000000	111.000000	62.000000	66.500000	24.220000	13.100000	40.000000	55.000000	19.000000	...
50%	2.000000	48.000000	121.000000	70.000000	78.200000	27.800000	14.100000	43.000000	67.000000	23.000000	...
75%	2.000000	63.000000	134.000000	78.000000	92.100000	32.100000	15.200000	45.000000	82.000000	27.000000	...
max	2.000000	85.000000	270.000000	132.000000	371.000000	130.210000	19.700000	57.000000	729.000000	1672.000000	...

8 rows × 25 columns

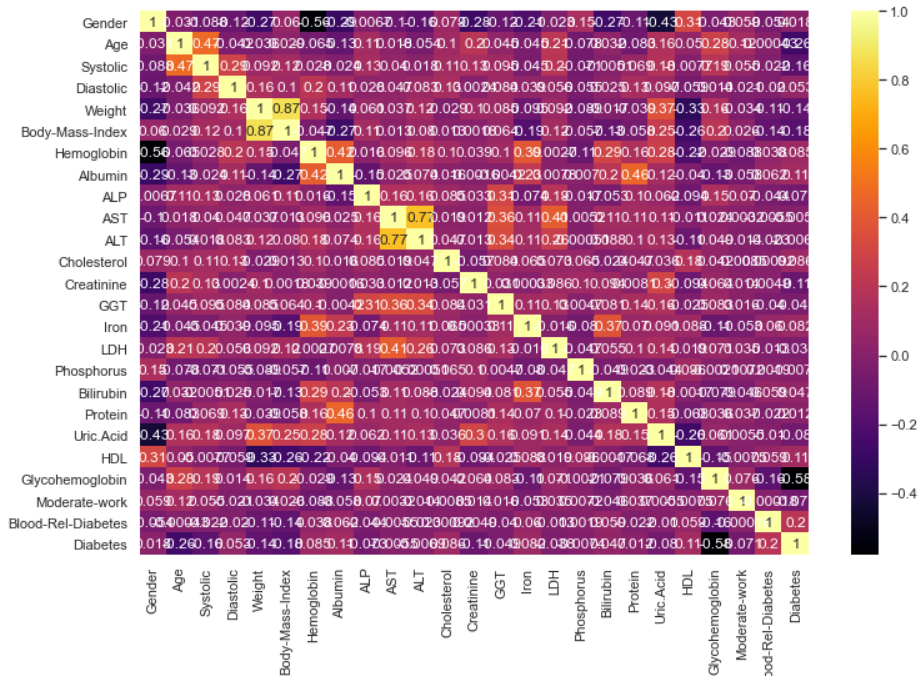
```
# Generate a horizontal bar plot of the counts of unique values in the 'Diabetes' column,
# sorted by index (in this case, assuming the index represents the unique values)
data["Diabetes"].value_counts().sort_index().plot.barh()
```

3]: <matplotlib.axes._subplots.AxesSubplot at 0x7f8c537d3430>



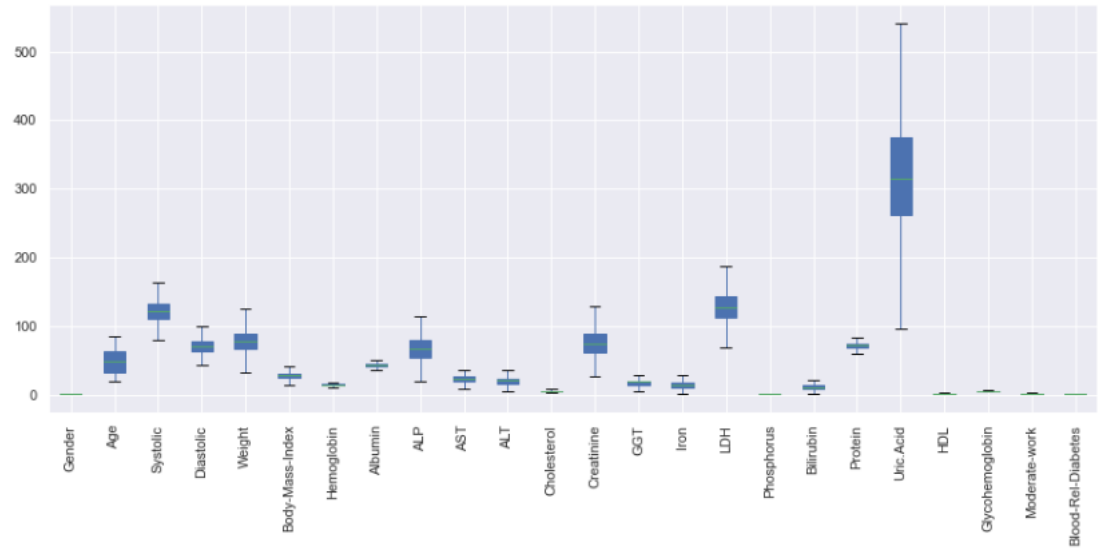
```
# Importing seaborn for statistical data visualization
import seaborn as sns
# Setting the size of the plot
plt.subplots(figsize=(12, 8))
# Creating a heatmap to visualize the correlation matrix of the DataFrame
sns.heatmap(data.corr(), cmap='inferno', annot=True)
```

31]: <matplotlib.axes._subplots.AxesSubplot at 0x7f8c5d410cd0>



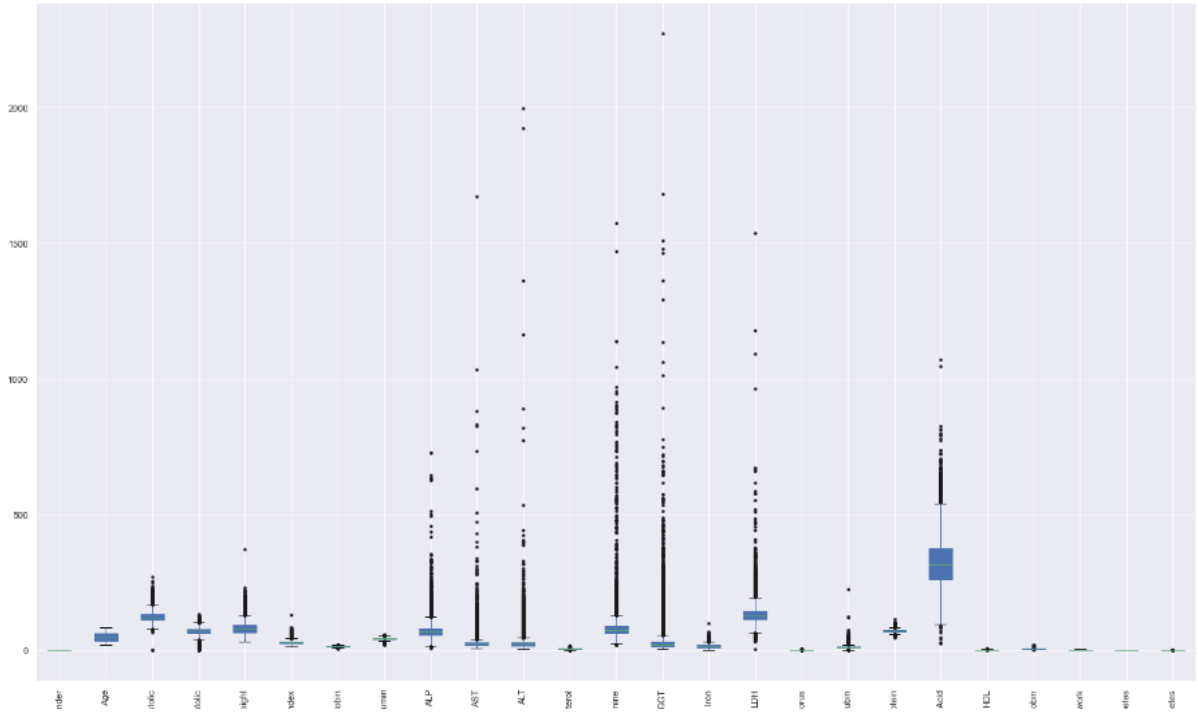

```
# Set plot size
plt.subplots(figsize=(15, 6))
# Create box plot with filled boxes and custom outlier style
X.boxplot(patch_artist=True, sym="k.")
# Rotate x-axis labels for readability
plt.xticks(rotation=90)
```

08]: (array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17,
        18, 19, 20, 21, 22, 23, 24]),
 <a list of 24 Text major ticklabel objects>)



```
# Set the size of the plot to be 25 inches in width and 15 inches in height
plt.subplots(figsize=(25, 15))
# Create a box plot of the DataFrame columns
# Using patch_artist=True to fill the box plot with color
# Setting sym="k." to customize the outlier style
data.boxplot(patch_artist=True, sym="k.")
# Rotate the x-axis labels by 90 degrees for better readability
plt.xticks(rotation=90)
```

```
)2]: (array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
        18, 19, 20, 21, 22, 23, 24, 25]),
 <a list of 25 Text major ticklabel objects>)
```



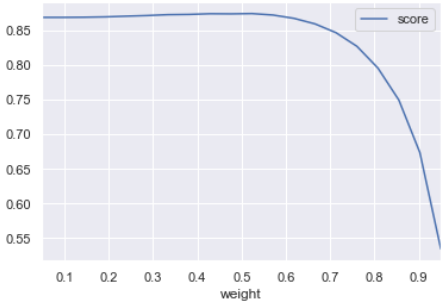
```
gsc = GridSearchCV(
    estimator=LogisticRegression(),  # Using Logistic Regression as the estimator
    param_grid={
        'class_weight': [{0: x, 1: 1.0 - x} for x in weights]  # Grid of class weights to search
    },
    scoring='accuracy',  # Evaluation metric
    cv=10  # 10-fold cross-validation
)

# Fit GridSearchCV to the data
grid_result = gsc.fit(X, y)

# Print the best parameters found by GridSearchCV
print("Best parameters : %s" % grid_result.best_params_)

# Plot the relationship between weights and accuracy score
dataz = pd.DataFrame({'score': grid_result.cv_results_['mean_test_score'],
                      'weight': weights})
dataz.plot(x='weight')  # Plot weights vs accuracy score
```

```
Best parameters : {'class_weight': {0: 0.5236842105263158, 1: 0.4763157894736842}}
```

Out[123]: <matplotlib.axes._subplots.AxesSubplot at 0x7f8c45e86760>

n [124]:

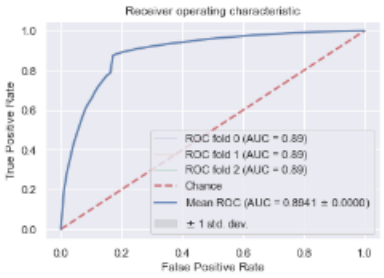
```
class_weight = {0: 0.5236842105263158,
                1: 0.4763157894736842}
```



SVM:

In [127]:

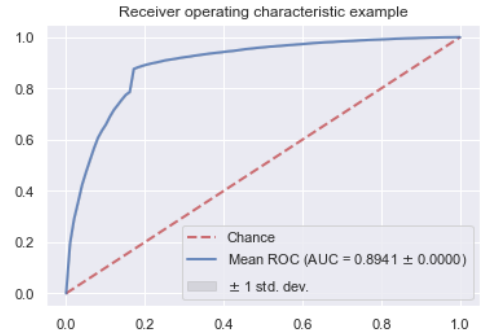
```
import numpy as np
import matplotlib.pyplot as plt

from sklearn import svm, datasets
from sklearn.metrics import auc
from sklearn.metrics import plot_roc_curve
from sklearn.model_selection import StratifiedKFold



# #############################################################################
# Classification and ROC analysis

# Run classifier with cross-validation and plot ROC curves
cv = ShuffleSplit(n_splits=3, test_size=0.2, random_state=42)
classifier = svm.SVC(kernel='rbf', probability=True, class_weight=class_weight,
                     random_state=42)

tprs = []
aucs = []
mean_fpr = np.linspace(0, 1, 100)

fig, ax = plt.subplots()
for i, (train, test) in enumerate(cv.split(X, y)):
    classifier.fit(X, y)
    viz = plot_roc_curve(classifier, X, y,
                         name='ROC fold {}'.format(i),
                         alpha=0.3, lw=1, ax=ax)
    interp_tpr = np.interp(mean_fpr, viz.fpr, viz.tpr)
    interp_tpr[0] = 0.0
    tprs.append(interp_tpr)
    aucs.append(viz.roc_auc)

ax.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r',
        label='Chance', alpha=.8)

mean_tpr = np.mean(tprs, axis=0)
mean_tpr[-1] = 1.0
mean_auc = auc(mean_fpr, mean_tpr)
std_auc = np.std(aucs)
ax.plot(mean_fpr, mean_tpr, color='b',
        label=r'Mean ROC (AUC = %0.4f $\pm$ %0.4f)' % (mean_auc, std_auc),
        lw=2, alpha=.8)

std_tpr = np.std(tprs, axis=0)
tprs_upper = np.minimum(mean_tpr + std_tpr, 1)
tprs_lower = np.maximum(mean_tpr - std_tpr, 0)
ax.fill_between(mean_fpr, tprs_lower, tprs_upper, color='grey', alpha=.2,
                label=r'$\pm$ 1 std. dev.')

ax.set(xlim=[-0.05, 1.05], ylim=[-0.05, 1.05],
       title="Receiver operating characteristic")
ax.legend(loc="lower right")
plt.show()
```



LR:

```
import numpy as np
import matplotlib.pyplot as plt

from sklearn import svm, datasets
from sklearn.metrics import auc
from sklearn.metrics import plot_roc_curve
from sklearn.model_selection import StratifiedKFold

# #############################################################################
# Data IO and generation

# Import some data to play with
#iris = datasets.load_iris()
#X = iris.data
#y = iris.target
#X, y = X[y != 2], y[y != 2]
#n_samples, n_features = X.shape

# Add noisy features
#random_state = np.random.RandomState(0)
#X = np.c_[X, random_state.randn(n_samples, 200 * n_features)]

# #############################################################################
# Classification and ROC analysis

# Run classifier with cross-validation and plot ROC curves
cv = ShuffleSplit(n_splits=3, test_size=0.2, random_state=0)
classifier1 = LogisticRegression(class_weight=class_weight,random_state=0)
tprs = []
aucs = []
mean_fpr = np.linspace(0, 1, 100)

fig, ax = plt.subplots()
for i, (train, test) in enumerate(cv.split(X, y)):
    classifier.fit(X, y)
    #viz = plot_roc_curve(classifier, X, y,
                         #  name='ROC fold {}'.format(i),
                           #alpha=0.3, lw=1, ax=ax)
    interp_tpr = np.interp(mean_fpr, viz.fpr, viz.tpr)
    interp_tpr[0] = 0.0
    tprs.append(interp_tpr)
    aucs.append(viz.roc_auc)
```

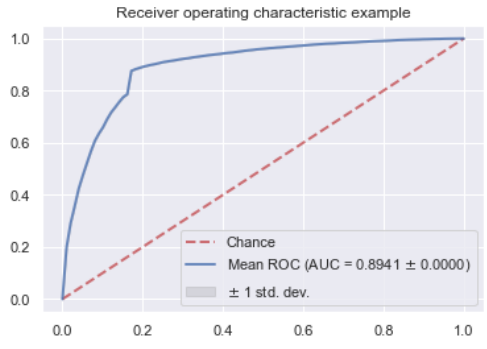
```
ax.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r',
        label='Chance', alpha=.8)

mean_tpr = np.mean(tprs, axis=0)
mean_tpr[-1] = 1
mean_auc = auc(mean_fpr, mean_tpr)
std_auc = np.std(aucs)
ax.plot(mean_fpr, mean_tpr, color='b',
        label=r'Mean ROC (AUC = %0.4f $\pm$ %0.4f)' % (mean_auc, std_auc),
        lw=2, alpha=.8)

std_tpr = np.std(tprs, axis=0)
tprs_upper = np.minimum(mean_tpr + std_tpr, 1)
tprs_lower = np.maximum(mean_tpr - std_tpr, 0)
ax.fill_between(mean_fpr, tprs_lower, tprs_upper, color='grey', alpha=.2,
                label=r'$\pm$ 1 std. dev.')

ax.set(xlim=[-0.05, 1.05], ylim=[-0.05, 1.05],
       title="Receiver operating characteristic example")
ax.legend(loc="lower right")
plt.show()
```



RF:

```
import numpy as np
import matplotlib.pyplot as plt

from sklearn import svm, datasets
from sklearn.metrics import auc
from sklearn.metrics import plot_roc_curve
from sklearn.model_selection import StratifiedKFold

# #############################################################################
# Data IO and generation

# Import some data to play with
#iris = datasets.load_iris()
#X = iris.data
#y = iris.target
#X, y = X[y != 2], y[y != 2]
#n_samples, n_features = X.shape

# Add noisy features
#random_state = np.random.RandomState(0)
#X = np.c_[X, random_state.randn(n_samples, 200 * n_features)]

# #############################################################################
# Classification and ROC analysis

# Run classifier with cross-validation and plot ROC curves
cv = ShuffleSplit(n_splits=3, test_size=0.2, random_state=42)
classifier = RandomForestClassifier(class_weight=class_weight,random_state=42)
tprs = []
aucs = []
mean_fpr = np.linspace(0, 1, 100)

fig, ax = plt.subplots()
for i, (train, test) in enumerate(cv.split(X, y)):
    classifier.fit(X, y)
    #viz = plot_roc_curve(classifier, X, y,
                         #  name='ROC fold {}'.format(i),
                          # alpha=0.3, lw=1, ax=ax)
    interp_tpr = np.interp(mean_fpr, viz.fpr, viz.tpr)
    interp_tpr[0] = 0.0
    tprs.append(interp_tpr)
    aucs.append(viz.roc_auc)
```

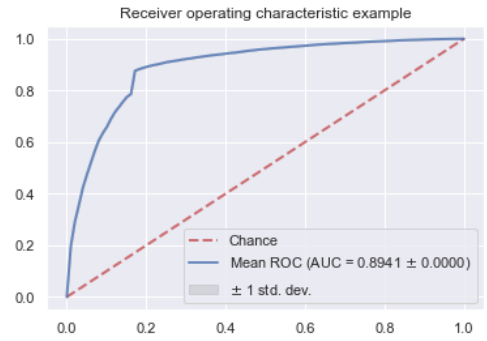
```
        label='Chance', alpha=.8)

mean_tpr = np.mean(tprs, axis=0)
mean_tpr[-1] = 1.0
mean_auc = auc(mean_fpr, mean_tpr)
std_auc = np.std(aucs)
ax.plot(mean_fpr, mean_tpr, color='b',
        label=r'Mean ROC (AUC = %0.4f $\pm$ %0.4f)' % (mean_auc, std_auc),
        lw=2, alpha=.8)

std_tpr = np.std(tprs, axis=0)
tprs_upper = np.minimum(mean_tpr + std_tpr, 1)
tprs_lower = np.maximum(mean_tpr - std_tpr, 0)
ax.fill_between(mean_fpr, tprs_lower, tprs_upper, color='grey', alpha=.2,
                label=r'$\pm$ 1 std. dev.')

ax.set(xlim=[-0.05, 1.05], ylim=[-0.05, 1.05],
       title="Receiver operating characteristic example")
ax.legend(loc="lower right")
plt.show()
```



DT:

```
import numpy as np
import matplotlib.pyplot as plt

from sklearn import svm, datasets
from sklearn.metrics import auc
from sklearn.metrics import plot_roc_curve
from sklearn.model_selection import StratifiedKFold

# #############################################################################
# Data IO and generation

# Import some data to play with
#iris = datasets.load_iris()
#X = iris.data
#y = iris.target
#X, y = X[y != 2], y[y != 2]
#n_samples, n_features = X.shape

# Add noisy features
#random_state = np.random.RandomState(0)
#X = np.c_[X, random_state.randn(n_samples, 200 * n_features)]

# #############################################################################
# Classification and ROC analysis

# Run classifier with cross-validation and plot ROC curves
cv = ShuffleSplit(n_splits=3, test_size=0.2, random_state=42)
classifier = DecisionTreeClassifier(class_weight=class_weight,random_state=42)
tprs = []
aucs = []
mean_fpr = np.linspace(0, 1, 100)

fig, ax = plt.subplots()
for i, (train, test) in enumerate(cv.split(X, y)):
    classifier.fit(X, y)
   # viz = plot_roc_curve(classifier, X, y,
                        #  name='ROC fold {}'.format(i),
                        #  alpha=0.3, lw=1, ax=ax)
    interp_tpr = np.interp(mean_fpr, viz.fpr, viz.tpr)
    interp_tpr[0] = 0.0
    tprs.append(interp_tpr)
    aucs.append(viz.roc_auc)
```

```
std_tpr = np.std(tprs, axis=0)
tprs_upper = np.minimum(mean_tpr + std_tpr, 1)
tprs_lower = np.maximum(mean_tpr - std_tpr, 0)
ax.fill_between(mean_fpr, tprs_lower, tprs_upper, color='grey', alpha=.2,
                label=r'$\pm$ 1 std. dev.')

ax.set(xlim=[-0.05, 1.05], ylim=[-0.05, 1.05],
       title="Receiver operating characteristic example")
ax.legend(loc="lower right")
plt.show()
```



ANN

```
import keras
from keras.models import Sequential
from keras.layers import Dense
```

```
Using TensorFlow backend.
```

```
classifier=Sequential()
classifier.add(Dense(units=32, kernel_initializer='uniform',activation='relu',input_dim=24))
classifier.add(Dense(units=6, kernel_initializer='uniform',activation='relu'))
classifier.add(Dense(units=1, kernel_initializer='uniform',activation='sigmoid'))
classifier.compile(optimizer='adam',loss="binary_crossentropy",metrics=['accuracy'])
classifier.fit(X_train,y_train,batch_size=10,epochs=100,class_weight=class_weight)
```

```
Epoch 1/100
29663/29663 [==============================] - 4s 120us/step - loss: 0.1562 - accuracy: 0.8740
Epoch 2/100
29663/29663 [==============================] - 4s 138us/step - loss: 0.1391 - accuracy: 0.8925
Epoch 3/100
29663/29663 [==============================] - 4s 150us/step - loss: 0.1309 - accuracy: 0.8986
Epoch 4/100
29663/29663 [==============================] - 4s 147us/step - loss: 0.1261 - accuracy: 0.8986
Epoch 5/100
29663/29663 [==============================] - 5s 154us/step - loss: 0.1235 - accuracy: 0.8998
Epoch 6/100
29663/29663 [==============================] - 4s 142us/step - loss: 0.1227 - accuracy: 0.9009
Epoch 7/100
29663/29663 [==============================] - 4s 151us/step - loss: 0.1219 - accuracy: 0.9013
Epoch 8/100
29663/29663 [==============================] - 4s 149us/step - loss: 0.1215 - accuracy: 0.9011
Epoch 9/100
29663/29663 [==============================] - 5s 178us/step - loss: 0.1209 - accuracy: 0.9020
Epoch 10/100
```

```
classifier.summary()
```

```
Model: "sequential_1"
```

```
#clf_svc_rbf.fit(X_train,y_train)
from sklearn.metrics import confusion_matrix,classification_report,roc_auc_score,auc,f1_score

y_pred = classifier.predict(X_test)>0.8


import matplotlib.pyplot as plt
cm = confusion_matrix(y_test,y_pred)

#plt.figure(figsize=(5,5))
#sns.heatmap(cm,annot=True)
#plt.show()

#print(classification_report(y_test,y_pred_clf_svc_rbf))

print(classification_report(y_test, y_pred))
#plot_confusion_matrix(confusion_matrix(y_test, y_pred))

from sklearn.metrics import roc_curve, auc
false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test, y_pred)
roc_auc = auc(false_positive_rate, true_positive_rate)
roc_auc
```

```
              precision    recall  f1-score   support

           0       0.46      0.73      0.56       951
           1       0.96      0.87      0.91      6465

    accuracy                           0.85      7416
   macro avg       0.71      0.80      0.74      7416
weighted avg       0.89      0.85      0.87      7416
```

```
]: 0.8010122450174562
```

```
from sklearn import datasets
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import VotingClassifier
```

```
clf1 = SVC(kernel='rbf', C=1, class_weight=class_weight,random_state=42)
clf2 = LogisticRegression(class_weight=class_weight,random_state=42)
clf3 = RandomForestClassifier(class_weight=class_weight,random_state=42)
clf4 = DecisionTreeClassifier(class_weight=class_weight,random_state=42)
#clf5 = Sequential()
```

```
eclf = VotingClassifier( estimators=[('svm', clf1), ('lr', clf2), ('rf', clf3), ('dt',clf4)],
     voting='hard')
```

```
for clf, label in zip([clf1, clf2, clf3,clf4 ,eclf], ['SVM', 'LR', 'RF','DT', 'Ensemble']):
    scores = cross_val_score(clf, X, y, scoring='accuracy', cv=3)
print("Accuracy: %0.4f (+/- %0.4f) [%s]" % (scores.mean(), scores.std(), label))
scores
```

```
Accuracy: 0.8888 (+/- 0.0028) [Ensemble]
```

```
5]: array([0.88495146, 0.89004854, 0.89149608])
```

```
```

Discussion or Analysis:

The Python code is used for data preprocessing, feature selection, class imbalance resolution, and model development.

Conclusions:

In summary, we have explored new avenues for improving predictive precision by using a multimodal machine learning methodology to address unbalanced data in diabetes prediction. By proactively correcting class imbalance using techniques like ensembles and deep learning, our study significantly strengthens the validity of clinical decision support systems in identifying diabetes risk. The foundation for future research and development into more dependable and effective predictive models for diabetes therapy is laid by these findings.

Appendix:

We are in the process of development of code and process related to it.

CODE:

```
# Importing necessary libraries
import numpy as np # For numerical operations.
import copy  # For creating deep copies of objects
from sklearn import preprocessing # For data preprocessing
import tensorflow as tf  # For building and training neural networks
from tensorflow import keras  # High-level neural networks API
import os  # For interacting with the operating system
import pandas as pd  # For data manipulation and analysis
from matplotlib import pyplot as plt  # For data visualization
from numpy.random import seed  # For seeding random number generators
np.random.seed(2095)  # Setting seed for reproducibility


data = pd.read_excel("C:\\Users\\thota\\OneDrive\\Desktop\\ML project 2024
spring\\CardiacPrediction.xlsx",engine="openpyxl")

data.drop(['SEQN','Annual-Family-Income','Height','Ratio-Family-Income-Poverty','X60-sec-pulse',
        'Health-Insurance','Glucose','Vigorous-work','Total-Cholesterol','CoronaryHeartDisease','Blood-Rel-
        Stroke','Red-Cell-Distribution-Width','Triglycerides','Mean-Platelet-Vol','Platelet-
        count','Lymphocyte','Monocyte','Eosinophils','Mean-cell-Hemoglobin','White-Blood-Cells','Red-Blood-
        Cells','Basophils','Mean-Cell-Vol','Mean-Cell-Hgb-Conc.','Hematocrit','Segmented-Neutrophils'], axis = 1,
        inplace=True)

# Replace instances of '3' with '1' in the 'Diabetes' column of the DataFrame
data['Diabetes'].loc[data['Diabetes'] == 3] = 1

data = data[['Gender', 'Age', 'Systolic', 'Diastolic', 'Weight', 'Body-Mass-Index',
      'Hemoglobin', 'Albumin', 'ALP', 'AST', 'ALT', 'Cholesterol',
      'Creatinine', 'GGT', 'Iron', 'LDH', 'Phosphorus',
      'Bilirubin', 'Protein', 'Uric.Acid', 'HDL',
      'Glycohemoglobin', 'Moderate-work',
      'Blood-Rel-Diabetes', 'Diabetes']]

# Generate a horizontal bar plot of the counts of unique values in the 'Diabetes' column,
# sorted by index (in this case, assuming the index represents the unique values)
data["Diabetes"].value_counts().sort_index().plot.barh()

# Importing seaborn for statistical data visualization
import seaborn as sns
# Setting the size of the plot
plt.subplots(figsize=(12, 8))
# Creating a heatmap to visualize the correlation matrix of the DataFrame
sns.heatmap(data.corr(), cmap='inferno', annot=True)

# Set the size of the plot to be 25 inches in width and 15 inches in height
plt.subplots(figsize=(25, 15))
# Create a box plot of the DataFrame columns
# Using patch_artist=True to fill the box plot with color
# Setting sym="k." to customize the outlier style
data.boxplot(patch_artist=True, sym="k.")
# Rotate the x-axis labels by 90 degrees for better readability
plt.xticks(rotation=90)
```

```
# Define a function to detect outliers in a feature
def detect_outlier(feature):
    # Calculate the first and third quartiles
    first_q = np.percentile(feature, 25)
    third_q = np.percentile(feature, 75)
    # Calculate the interquartile range (IQR)
    IQR = third_q - first_q
    # Define the lower and upper bounds for outlier detection
    lower_bound = first_q - (IQR * 1.5)
    upper_bound = third_q + (IQR * 1.5)
    # Initialize flag to indicate if outliers are detected
    flag = False
    # Check if there are outliers below the lower bound or above the upper bound
    if lower_bound > np.min(feature) or upper_bound < np.max(feature):
        flag = True
    return flag

def  remove_outlier(feature):
    first_q = np.percentile(X[feature], 25)
    third_q = np.percentile(X[feature], 75)
    IQR = third_q - first_q
    IQR *= 1.5

    minimum = first_q - IQR # the acceptable minimum value
    maximum = third_q + IQR # the acceptable maximum value

    median = X[feature].median()

    """
    # any value beyond the acceptance range are considered
    as outliers.
    # we replace the outliers with the median value of that
      feature.
    """

    X.loc[X[feature] < minimum, feature] = median
    X.loc[X[feature] > maximum, feature] = median

# taking all the columns except the last one
# last column is the label

X = data.iloc[:, :-1]
for i in range(len(X.columns)):
        remove_outlier(X.columns[i])

# Iterate through the columns of the DataFrame
for i in range(len(X.columns)):
    # Check if the column contains outliers using the detect_outlier function
    if detect_outlier(X[X.columns[i]]):
        # Print the column name if outliers are detected
        print(X.columns[i], "Contains Outlier")

# Set plot size
plt.subplots(figsize=(15, 6))
# Create box plot with filled boxes and custom outlier style
X.boxplot(patch_artist=True, sym="k.")
# Rotate x-axis labels for readability
plt.xticks(rotation=90)
```

```
# Importing necessary libraries
import numpy as np  # For numerical operations
import pandas as pd  # For data manipulation and analysis
import matplotlib.pyplot as plt  # For data visualization
%matplotlib inline  # Display matplotlib plots inline

import seaborn as sns  # For statistical data visualization
sns.set()  # Set seaborn default style
from sklearn.preprocessing import MinMaxScaler, StandardScaler, LabelEncoder  # For data preprocessing
from sklearn.svm import SVC  # Support Vector Classifier
from sklearn.naive_bayes import GaussianNB  # Gaussian Naive Bayes classifier
from sklearn.linear_model import LogisticRegression  # Logistic Regression classifier
# from xgboost import XGBClassifier, plot_importance  # XGBoost classifier (uncomment if needed)
from sklearn.model_selection import train_test_split  # For splitting dataset into train and test sets
from sklearn.metrics import accuracy_score, confusion_matrix  # For model evaluation

# Initialize StandardScaler for feature scaling
scaler = StandardScaler()
# Scale the features using StandardScaler
scaled_data = scaler.fit_transform(X)
# Create a DataFrame from the scaled data with column names
scaled_df = pd.DataFrame(data=scaled_data, columns=X.columns)
# Display the first few rows of the scaled DataFrame
scaled_df.head()

# Importing necessary libraries
import sklearn.feature_selection as fs  # For feature selection
import matplotlib.pyplot as plt  # For data visualization

# Initialize SelectKBest with k='all'
df2 = fs.SelectKBest(k='all')
# Fit SelectKBest to the data
df2.fit(X, y)

# Get the names of selected features
names = X.columns.values[df2.get_support()]
# Get the scores of selected features
scores = df2.scores_[df2.get_support()]

# Combine feature names and their corresponding scores
names_scores = list(zip(names, scores))
# Create a DataFrame from feature names and their scores
ns_df = pd.DataFrame(data=names_scores, columns=['Features', 'F_Scores'])

# Sort the DataFrame by F-scores in descending order
ns_df_sorted = ns_df.sort_values(['F_Scores', 'Features'], ascending=[False, True])

# Print the sorted DataFrame
print(ns_df_sorted)




# Import GridSearchCV for hyperparameter tuning
from sklearn.model_selection import GridSearchCV

# Define weights for class balancing
weights = np.linspace(0.05, 0.95, 20)
```

```
# Initialize GridSearchCV
gsc = GridSearchCV(
    estimator=LogisticRegression(),  # Using Logistic Regression as the estimator
    param_grid={
        'class_weight': [{0: x, 1: 1.0 - x} for x in weights]  # Grid of class weights to search
    },
    scoring='accuracy',  # Evaluation metric
    cv=10  # 10-fold cross-validation
)

# Fit GridSearchCV to the data
grid_result = gsc.fit(X, y)

# Print the best parameters found by GridSearchCV
print("Best parameters : %s" % grid_result.best_params_)

# Plot the relationship between weights and accuracy score
dataz = pd.DataFrame({'score': grid_result.cv_results_['mean_test_score'],
                      'weight': weights})
dataz.plot(x='weight')  # Plot weights vs accuracy score



from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, classification_report
from mlxtend.plotting import plot_decision_regions, plot_confusion_matrix
from matplotlib import pyplot as plt
lr = LogisticRegression(class_weight='balanced',random_state=420)

# Fit..
lr.fit(X_train, y_train)

# Predict..
y_pred = lr.predict(X_test)

# Evaluate the model
print(classification_report(y_test, y_pred))
plot_confusion_matrix(confusion_matrix(y_test, y_pred))
from sklearn.metrics import roc_curve, auc
false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test, y_pred)
roc_auc = auc(false_positive_rate, true_positive_rate)
roc_auc



from sklearn.svm import SVC

clf_svc_rbf = SVC(kernel="rbf",class_weight='balanced',random_state=4200)
clf_svc_rbf.fit(X_train,y_train)
y_pred_clf_svc_rbf = clf_svc_rbf.predict(X_test)


import matplotlib.pyplot as plt
cm = confusion_matrix(y_test,y_pred_clf_svc_rbf)

#plt.figure(figsize=(5,5))
#sns.heatmap(cm,annot=True)
#plt.show()

#print(classification_report(y_test,y_pred_clf_svc_rbf))
```

```
print(classification_report(y_test, y_pred_clf_svc_rbf))
plot_confusion_matrix(confusion_matrix(y_test, y_pred_clf_svc_rbf))

from sklearn.metrics import roc_curve, auc
false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test, y_pred_clf_svc_rbf)
roc_auc = auc(false_positive_rate, true_positive_rate)
roc_auc



from sklearn.ensemble import RandomForestClassifier

rd = RandomForestClassifier(class_weight='balanced',random_state=4200)
rd.fit(X_train,y_train)
y_pred_rd = rd.predict(X_test)


import matplotlib.pyplot as plt
cm = confusion_matrix(y_test,y_pred_rd)

#plt.figure(figsize=(5,5))
#sns.heatmap(cm,annot=True,linewidths=.3)
#plt.show()

print(classification_report(y_test,y_pred_rd))
plot_confusion_matrix(confusion_matrix(y_test, y_pred_rd))

from sklearn.metrics import roc_curve, auc
false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test, y_pred_rd)
roc_auc = auc(false_positive_rate, true_positive_rate)
roc_auc
```

SVM:

```
evaluate a logistic regression model using k-fold cross-validation
from numpy import mean
from numpy import std
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import ShuffleSplit

from sklearn.linear_model import LogisticRegression
# create dataset
#X, y = make_classification(n_samples=1000, n_features=20, n_informative=15, n_redundant=5, random_state=1)
# prepare the cross-validation procedure
#cv = KFold(n_splits=5, test_size= 0.2, random_state=0)
cv = ShuffleSplit(n_splits=3, test_size=0.2, random_state=42)
# create model
model = SVC(kernel='rbf', C=1, class_weight=class_weight)
# evaluate model
scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1)
# report performance
print('Accuracy: %.4f (%.4f)' % (mean(scores), std(scores)))
scores
```

```
import numpy as np
import matplotlib.pyplot as plt

from sklearn import svm, datasets
from sklearn.metrics import auc
from sklearn.metrics import plot_roc_curve
from sklearn.model_selection import StratifiedKFold



# #############################################################################
# Classification and ROC analysis

# Run classifier with cross-validation and plot ROC curves
cv = ShuffleSplit(n_splits=3, test_size=0.2, random_state=42)
classifier = svm.SVC(kernel='rbf', probability=True, class_weight=class_weight,
                     random_state=42)

tprs = []
aucs = []
mean_fpr = np.linspace(0, 1, 100)

fig, ax = plt.subplots()
for i, (train, test) in enumerate(cv.split(X, y)):
    classifier.fit(X, y)
    viz = plot_roc_curve(classifier, X, y,
                         name='ROC fold {}'.format(i),
                         alpha=0.3, lw=1, ax=ax)
    interp_tpr = np.interp(mean_fpr, viz.fpr, viz.tpr)
    interp_tpr[0] = 0.0
    tprs.append(interp_tpr)
    aucs.append(viz.roc_auc)

ax.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r',
        label='Chance', alpha=.8)

mean_tpr = np.mean(tprs, axis=0)
mean_tpr[-1] = 1.0
mean_auc = auc(mean_fpr, mean_tpr)
std_auc = np.std(aucs)
ax.plot(mean_fpr, mean_tpr, color='b',
        label=r'Mean ROC (AUC = %0.4f $\pm$ %0.4f)' % (mean_auc, std_auc),
        lw=2, alpha=.8)

std_tpr = np.std(tprs, axis=0)
tprs_upper = np.minimum(mean_tpr + std_tpr, 1)
tprs_lower = np.maximum(mean_tpr - std_tpr, 0)
ax.fill_between(mean_fpr, tprs_lower, tprs_upper, color='grey', alpha=.2,
                label=r'$\pm$ 1 std. dev.')

ax.set(xlim=[-0.05, 1.05], ylim=[-0.05, 1.05],
       title="Receiver operating characteristic")
ax.legend(loc="lower right")
plt.show()
```

LR:

```
from numpy import mean
from numpy import std
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import ShuffleSplit

from sklearn.linear_model import LogisticRegression
# create dataset
#X, y = make_classification(n_samples=1000, n_features=20, n_informative=15, n_redundant=5, random_state=1)
# prepare the cross-validation procedure
#cv = KFold(n_splits=5, test_size= 0.2, random_state=0)
cv = ShuffleSplit(n_splits=3, test_size=0.2, random_state=42)
# create model
model = LogisticRegression(class_weight=class_weight)
# evaluate model
scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1)
# report performance
print('Accuracy: %.4f (%.4f)' % (mean(scores), std(scores)))
scores


import numpy as np
import matplotlib.pyplot as plt

from sklearn import svm, datasets
from sklearn.metrics import auc
from sklearn.metrics import plot_roc_curve
from sklearn.model_selection import StratifiedKFold

# #############################################################################
# Data IO and generation

# Import some data to play with
#iris = datasets.load_iris()
#X = iris.data
#y = iris.target
#X, y = X[y != 2], y[y != 2]
#n_samples, n_features = X.shape

# Add noisy features
#random_state = np.random.RandomState(0)
#X = np.c_[X, random_state.randn(n_samples, 200 * n_features)]

# #############################################################################
# Classification and ROC analysis

# Run classifier with cross-validation and plot ROC curves
cv = ShuffleSplit(n_splits=3, test_size=0.2, random_state=0)
classifier1 = LogisticRegression(class_weight=class_weight,random_state=0)
tprs = []
aucs = []
mean_fpr = np.linspace(0, 1, 100)

fig, ax = plt.subplots()
for i, (train, test) in enumerate(cv.split(X, y)):
    classifier.fit(X, y)
    #viz = plot_roc_curve(classifier, X, y,
                         #  name='ROC fold {}'.format(i),
                          #alpha=0.3, lw=1, ax=ax)
```

```
    interp_tpr = np.interp(mean_fpr, viz.fpr, viz.tpr)
    interp_tpr[0] = 0.0
    tprs.append(interp_tpr)
    aucs.append(viz.roc_auc)

ax.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r',
        label='Chance', alpha=.8)

mean_tpr = np.mean(tprs, axis=0)
mean_tpr[-1] = 1
mean_auc = auc(mean_fpr, mean_tpr)
std_auc = np.std(aucs)
ax.plot(mean_fpr, mean_tpr, color='b',
        label=r'Mean ROC (AUC = %0.4f $\pm$ %0.4f)' % (mean_auc, std_auc),
        lw=2, alpha=.8)

std_tpr = np.std(tprs, axis=0)
tprs_upper = np.minimum(mean_tpr + std_tpr, 1)
tprs_lower = np.maximum(mean_tpr - std_tpr, 0)
ax.fill_between(mean_fpr, tprs_lower, tprs_upper, color='grey', alpha=.2,
                label=r'$\pm$ 1 std. dev.')

ax.set(xlim=[-0.05, 1.05], ylim=[-0.05, 1.05],
       title="Receiver operating characteristic example")
ax.legend(loc="lower right")
plt.show()
```

RF:

```
from numpy import mean
from numpy import std
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import ShuffleSplit
from sklearn.ensemble import RandomForestClassifier

# create dataset
#X, y = make_classification(n_samples=1000, n_features=20, n_informative=15, n_redundant=5, random_state=1)
# prepare the cross-validation procedure
#cv = KFold(n_splits=5, test_size= 0.2, random_state=0)
cv = ShuffleSplit(n_splits=3, test_size=0.2, random_state=42)
# create model
model = RandomForestClassifier(class_weight=class_weight)
# evaluate model
scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1)
# report performance
print('Accuracy: %.4f (%.4f)' % (mean(scores), std(scores)))
scores

import numpy as np
import matplotlib.pyplot as plt

from sklearn import svm, datasets
from sklearn.metrics import auc
from sklearn.metrics import plot_roc_curve
from sklearn.model_selection import StratifiedKFold

# #############################################################################
# Data IO and generation
```

```
# Import some data to play with
#iris = datasets.load_iris()
#X = iris.data
#y = iris.target
#X, y = X[y != 2], y[y != 2]
#n_samples, n_features = X.shape

# Add noisy features
#random_state = np.random.RandomState(0)
#X = np.c_[X, random_state.randn(n_samples, 200 * n_features)]

# ############################################################################
# Classification and ROC analysis

# Run classifier with cross-validation and plot ROC curves
cv = ShuffleSplit(n_splits=3, test_size=0.2, random_state=42)
classifier = RandomForestClassifier(class_weight=class_weight,random_state=42)
tprs = []
aucs = []
mean_fpr = np.linspace(0, 1, 100)

fig, ax = plt.subplots()
for i, (train, test) in enumerate(cv.split(X, y)):
    classifier.fit(X, y)
    #viz = plot_roc_curve(classifier, X, y,
                         #  name='ROC fold {}'.format(i),
                          # alpha=0.3, lw=1, ax=ax)
    interp_tpr = np.interp(mean_fpr, viz.fpr, viz.tpr)
    interp_tpr[0] = 0.0
    tprs.append(interp_tpr)
    aucs.append(viz.roc_auc)

ax.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r',
        label='Chance', alpha=.8)

mean_tpr = np.mean(tprs, axis=0)
mean_tpr[-1] = 1.0
mean_auc = auc(mean_fpr, mean_tpr)
std_auc = np.std(aucs)
ax.plot(mean_fpr, mean_tpr, color='b',
        label=r'Mean ROC (AUC = %0.4f $\pm$ %0.4f)' % (mean_auc, std_auc),
        lw=2, alpha=.8)

std_tpr = np.std(tprs, axis=0)
tprs_upper = np.minimum(mean_tpr + std_tpr, 1)
tprs_lower = np.maximum(mean_tpr - std_tpr, 0)
ax.fill_between(mean_fpr, tprs_lower, tprs_upper, color='grey', alpha=.2,
                label=r'$\pm$ 1 std. dev.')

ax.set(xlim=[-0.05, 1.05], ylim=[-0.05, 1.05],
       title="Receiver operating characteristic example")
ax.legend(loc="lower right")
plt.show()
```

DT:

```
from numpy import mean
from numpy import std
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import ShuffleSplit
from sklearn.tree import DecisionTreeClassifier


# create dataset
#X, y = make_classification(n_samples=1000, n_features=20, n_informative=15, n_redundant=5, random_state=1)
# prepare the cross-validation procedure
#cv = KFold(n_splits=5, test_size= 0.2, random_state=0)
cv = ShuffleSplit(n_splits=3, test_size=0.2, random_state=42)
# create model
model = DecisionTreeClassifier(class_weight=class_weight)
# evaluate model
scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1)
# report performance
print('Accuracy: %.4f (%.4f)' % (mean(scores), std(scores)))
scores

import numpy as np
import matplotlib.pyplot as plt

from sklearn import svm, datasets
from sklearn.metrics import auc
from sklearn.metrics import plot_roc_curve
from sklearn.model_selection import StratifiedKFold

# #############################################################################
# Data IO and generation

# Import some data to play with
#iris = datasets.load_iris()
#X = iris.data
#y = iris.target
#X, y = X[y != 2], y[y != 2]
#n_samples, n_features = X.shape

# Add noisy features
#random_state = np.random.RandomState(0)
#X = np.c_[X, random_state.randn(n_samples, 200 * n_features)]

# #############################################################################
# Classification and ROC analysis

# Run classifier with cross-validation and plot ROC curves
cv = ShuffleSplit(n_splits=3, test_size=0.2, random_state=42)
classifier = DecisionTreeClassifier(class_weight=class_weight,random_state=42)
tprs = []
aucs = []
mean_fpr = np.linspace(0, 1, 100)

fig, ax = plt.subplots()
for i, (train, test) in enumerate(cv.split(X, y)):
    classifier.fit(X, y)
   # viz = plot_roc_curve(classifier, X, y,
                        #  name='ROC fold {}'.format(i),
```

```
                    #  alpha=0.3, lw=1, ax=ax)
    interp_tpr = np.interp(mean_fpr, viz.fpr, viz.tpr)
    interp_tpr[0] = 0.0
    tprs.append(interp_tpr)
    aucs.append(viz.roc_auc)

ax.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r',
        label='Chance', alpha=.8)

mean_tpr = np.mean(tprs, axis=0)
mean_tpr[-1] = 1.0
mean_auc = auc(mean_fpr, mean_tpr)
std_auc = np.std(aucs)
ax.plot(mean_fpr, mean_tpr, color='b',
        label=r'Mean ROC (AUC = %0.4f $\pm$ %0.4f)' % (mean_auc, std_auc),
        lw=2, alpha=.8)

std_tpr = np.std(tprs, axis=0)
tprs_upper = np.minimum(mean_tpr + std_tpr, 1)
tprs_lower = np.maximum(mean_tpr - std_tpr, 0)
ax.fill_between(mean_fpr, tprs_lower, tprs_upper, color='grey', alpha=.2,
                label=r'$\pm$ 1 std. dev.')

ax.set(xlim=[-0.05, 1.05], ylim=[-0.05, 1.05],
       title="Receiver operating characteristic example")
ax.legend(loc="lower right")
plt.show()
```

ANN:

```
import keras
from keras.models import Sequential
from keras.layers import Dense

#clf_svc_rbf.fit(X_train,y_train)
from sklearn.metrics import confusion_matrix,classification_report,roc_auc_score,auc,f1_score

y_pred = classifier.predict(X_test)>0.8


import matplotlib.pyplot as plt
cm = confusion_matrix(y_test,y_pred)

#plt.figure(figsize=(5,5))
#sns.heatmap(cm,annot=True)
#plt.show()

#print(classification_report(y_test,y_pred_clf_svc_rbf))

print(classification_report(y_test, y_pred))
#plot_confusion_matrix(confusion_matrix(y_test, y_pred))

from sklearn.metrics import roc_curve, auc
false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test, y_pred)
roc_auc = auc(false_positive_rate, true_positive_rate)
roc_auc
```

```
from sklearn import datasets
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import VotingClassifier

clf1 = SVC(kernel='rbf', C=1, class_weight=class_weight,random_state=42)
clf2 = LogisticRegression(class_weight=class_weight,random_state=42)
clf3 = RandomForestClassifier(class_weight=class_weight,random_state=42)
clf4 = DecisionTreeClassifier(class_weight=class_weight,random_state=42)
#clf5 = Sequential()

eclf = VotingClassifier( estimators=[('svm', clf1), ('lr', clf2), ('rf', clf3), ('dt',clf4)],
    voting='hard')

for clf, label in zip([clf1, clf2, clf3,clf4 ,eclf], ['SVM', 'LR', 'RF','DT', 'Ensemble']):
   scores = cross_val_score(clf, X, y, scoring='accuracy', cv=3)
print("Accuracy: %0.4f (+/- %0.4f) [%s]" % (scores.mean(), scores.std(), label))
scores
```