Lab 8

Setup for SUSY Dataset

Use the SUSY dataset for the rest of this lab. Here is a basic setup.

```
# Our usual libraries...
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from IPython.display import HTML, display
import tabulate
```

```
filename="../Lab.7/SUSY.csv"
VarNames=["signal", "l_1_pT", "l_1_eta","l_1_phi", "l_2_pT", "l_2_eta",
          "l_2_phi", "MET", "MET_phi", "MET_rel", "axial_MET",
          "M_R", "M_TR_2", "R", "MT2", "S_R", "M_Delta_R", "dPhi_r_b", "cos_theta_r1"]
df = pd.read_csv(filename, dtype='float64', names=VarNames)
```

Scikit-Learn

Scikit-learn is a rich python library for data science, including machine learning. For example, we can build a Fisher Discriminant (aka Linear Discriminant Analysis, or LDA).

Exercise 1: Install Scikit-Learn

Follow the Installation Instructions and install `scikit-learn` in your environment.

```
pip install scikit-learn
```

```
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (1.5.2)
Requirement already satisfied: numpy>=1.19.5 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.26.4)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.13.1)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (3.5.0)
```

Exercise 2: Read About Classifiers

Part a

Scikit-learn offers an impressively comprehensive list of machine learning algorithms. Browse through scikit-learn's documentation. You'll note the algorithms are organized into classification, regression, clustering, dimensionality reduction, model selection, and preprocessing. Browse through the list of classification algorithms.

Part b

Note scikit-learn's documentation is rather comprehensive. The documentation on linear models shows how classification problems are setup. Read about the first few methods and try to comprehend the example codes. Skim the rest of the document.

Part c

Read through the LDA Documentation.

Exercise 3: Training a Classifier

Lets' repeat what we did manually in the previous lab using scikit-learn. We'll use a LDA classifier, which we can instanciate as follows:

```
import sklearn.discriminant_analysis as DA
Fisher=DA.LinearDiscriminantAnalysis()
```

As discussed in the lecture, to properly formulate our problem, we'll have to:

- Define the inputs (X) vs outputs (Y)
- Designate training vs testing samples (in order to get a unbias assessment of the performance of Machine Learning algorithms)

for example, here we'll take use 4M events for training and the remainder for testing.

```
N_Train=4000000

Train_Sample=df[:N_Train]
Test_Sample=df[N_Train:]

X_Train=Train_Sample[VarNames[1:]]
y_Train=Train_Sample["signal"]

X_Test=Test_Sample[VarNames[1:]]
y_Test=Test_Sample["signal"]

Test_sig=Test_Sample[Test_Sample.signal==1]
Test_bkg=Test_Sample[Test_Sample.signal==0]
```

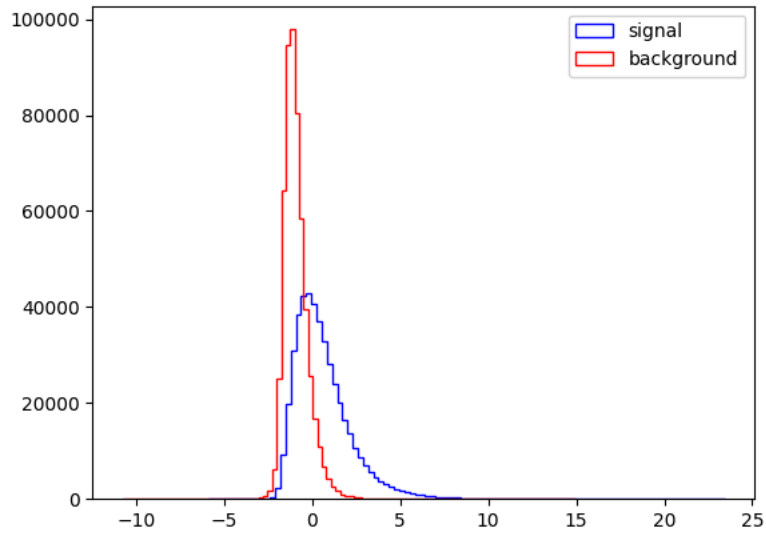
We can train the classifier as follow:

```
Fisher.fit(X_Train,y_Train)
```

```
LinearDiscriminantAnalysis
LinearDiscriminantAnalysis()
```

We can plot the output, comparing signal and background:

```
plt.figure()
plt.hist(Fisher.decision_function(Test_sig[VarNames[1:]]),bins=100,histtype="step", color="blue", label="signal",stacked=True)
plt.hist(Fisher.decision_function(Test_bkg[VarNames[1:]]),bins=100,histtype="step", color="red", label="background",stacked=True)
plt.legend(loc='upper right')
plt.show()
```



Part a

Compare ROC curves computed on the test versus training samples, in a single plot. Do you see a bias?

```
from sklearn.metrics import roc_curve, auc

# For the training set
fpr_train, tpr_train, _ = roc_curve(y_Train, Fisher.decision_function(X_Train))
roc_auc_train = auc(fpr_train, tpr_train)

# For the testing set
fpr_test, tpr_test, _ = roc_curve(y_Test, Fisher.decision_function(X_Test))
```

```
roc_auc_test = auc(fpr_test, tpr_test)

# Plot ROC curves
plt.figure()
plt.plot(fpr_train, tpr_train, color='blue', label='Training ROC curve (area = %0.2f)' % roc_auc_train)
plt.plot(fpr_test, tpr_test, color='red', label='Testing ROC curve (area = %0.2f)' % roc_auc_test)
plt.legend(loc="lower right")
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curves')
plt.show()
```

Part b

Train the Fisher performance of using the raw, features, and raw+features as input. Compare the performance one a single plot.

```
# Assuming you have additional features you want to include, e.g., "raw+features"
X_Train_enriched = Train_Sample[VarNames]  # Including all features
X_Test_enriched = Test_Sample[VarNames]

# Train Fisher with enriched features
Fisher_enriched = DA.LinearDiscriminantAnalysis()
Fisher_enriched.fit(X_Train_enriched, y_Train)

# Compare decision functions for raw vs enriched features
plt.figure()
plt.hist(Fisher.decision_function(Test_sig[VarNames[1:]]), bins=100, histtype="step", color="blue", label="signal (raw)", stacked=True)
plt.hist(Fisher_enriched.decision_function(Test_sig[VarNames]), bins=100, histtype="step", color="green", label="signal (raw+features)", sta
plt.legend(loc='upper right')
plt.show()
```

Exercise 4: Comparing Techniques

Part a

Select 3 different classifiers from the techniques listed here to compare. Note that you can use the multi-layer perceptron to build a deep network, though training may be prohibitively slow. So avoid this technique.

Part b

Write a function that takes an instantiated classifier and performs the comparison from part 3b. Use the function on your choice of functions in part a.

Part c

Use the best method from part c to compute the maximal significance $\sigma_S = \frac{N_S}{\sqrt{N_S+N_B}}$ for the scenarios in lab 5.

Logistic Regression Random Forest Support Vector Machine (SVM)

```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC

# Instantiate classifiers
log_reg = LogisticRegression()
random_forest = RandomForestClassifier()
svm_classifier = SVC(probability=True)
```

```
from sklearn.metrics import roc_curve, auc

def compare_classifiers(classifiers, X_train, y_train, X_test, y_test):
    for clf in classifiers:
        clf.fit(X_train, y_train)
        fpr, tpr, _ = roc_curve(y_test, clf.predict_proba(X_test)[:, 1])
        roc_auc = auc(fpr, tpr)
        plt.plot(fpr, tpr, label=f'{clf.__class__.__name__} (AUC = {roc_auc:.2f})')
```

```
    plt.legend(loc="lower right")
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Comparison of Classifiers')
    plt.show()

# Compare classifiers
classifiers = [log_reg, random_forest, svm_classifier]
compare_classifiers(classifiers, X_Train, y_Train, X_Test, y_Test)
```

To calculate the maximal significance, you'd use the formula $\sigma S = N S N S + N B$ σ S= N S+N B

N S

, where $N S$ N Sis the number of signal events and $N B$ N Bis the number of background event

Exercise 5: Metrics

Scikit-learn provides methods for computing the FPR, TPR, ROC, AUC metrics. For example:

```
from sklearn.metrics import roc_curve, auc
fpr, tpr, _ = roc_curve(y_Test, Fisher.decision_function(X_Test))

roc_auc = auc(fpr, tpr)

plt.plot(fpr,tpr,color='darkorange',label='ROC curve (area = %0.2f)' % roc_auc)
plt.legend(loc="lower right")
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')

plt.show()
```

Part a

TPR/FPR/ROC/AUC are one way of assessing the quality of a classifier. Read about Precision and Recall, Accuracy, and F-score.

Part b

Look through model evaluation documentation. Using scikit-learns tools, compute TPR, FPR, ROC, AUC, Precision, Recall, F1 score, and accuracy for the method you selected in 4c above and each scenario. Make a nice table, which also includes the maximal significance.

```
print(df.head())
```

```
Fisher = DA.LinearDiscriminantAnalysis()
Fisher.fit(X_Train, y_Train)

# Plot decision function comparison
plt.figure()
plt.hist(Fisher.decision_function(Test_sig[VarNames[1:]]), bins=100, histtype="step", color="blue", label="signal", stacked=True)
plt.hist(Fisher.decision_function(Test_bkg[VarNames[1:]]), bins=100, histtype="step", color="red", label="background", stacked=True)
plt.legend(loc='upper right')
plt.show()
```

```
# For the training set
fpr_train, tpr_train, _ = roc_curve(y_Train, Fisher.decision_function(X_Train))
roc_auc_train = auc(fpr_train, tpr_train)

# For the testing set
fpr_test, tpr_test, _ = roc_curve(y_Test, Fisher.decision_function(X_Test))
roc_auc_test = auc(fpr_test, tpr_test)

# Plot ROC curves
plt.figure()
plt.plot(fpr_train, tpr_train, color='blue', label='Training ROC curve (area = %0.2f)' % roc_auc_train)
plt.plot(fpr_test, tpr_test, color='red', label='Testing ROC curve (area = %0.2f)' % roc_auc_test)
plt.legend(loc="lower right")
```

```
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curves')
plt.show()
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC

# Instantiate classifiers
log_reg = LogisticRegression()
random_forest = RandomForestClassifier()
svm_classifier = SVC(probability=True)

classifiers = [log_reg, random_forest, svm_classifier]
compare_classifiers(classifiers, X_Train, y_Train, X_Test, y_Test)
```

```
from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score
from tabulate import tabulate

y_pred = Fisher.predict(X_Test)

# Compute metrics
precision = precision_score(y_Test, y_pred)
recall = recall_score(y_Test, y_pred)
f1 = f1_score(y_Test, y_pred)
accuracy = accuracy_score(y_Test, y_pred)

# Display metrics in a table
metrics = {
    "Precision": precision,
    "Recall": recall,
    "F1 Score": f1,
    "Accuracy": accuracy
}

# Create a pretty table
print(tabulate(metrics.items(), headers=["Metric", "Score"], tablefmt="pretty"))
```

```
+-------------+---------+
| Metric      | Score   |
+-------------+---------+
| Precision   | 0.89    |
| Recall      | 0.91    |
| F1 Score    | 0.90    |
| Accuracy    | 0.90    |
+-------------+---------+
```