

HW2

Dor & Guy

Theory Questions

Q1: To evaluate how well our model performs at T1D classification, we need to have evaluation metrics that measures of its performances/accuracy. Which evaluation metric is more important to us: model accuracy or model performance? Give a simple example that illustrates your claim.

Q2: T1D is often associated with other comorbidities such as a heart attack. You are asked to design a ML algorithm to predict which patients are going to suffer a heart attack. Relevant patient features for the algorithm may include blood pressure (BP), body-mass index (BMI), age (A), level of physical activity (P), and income (I). You should choose between two classifiers: the first uses only BP and BMI features and the other one uses all of the features available to you. Explain the pros and cons of each choice.

Q3: A histologist wants to use machine learning to tell the difference between pancreas biopsies that show signs of T1D and those that do not. She has already come up with dozens of measurements to take, such as color, size, uniformity and cell-count, but she isn't sure which model to use. The biopsies are really similar, and it is difficult to distinguish them from the human eye, or by just looking at the features. Which of the following is better: logistic regression, linear SVM or nonlinear SVM? Explain your answer.

Q4: What are the differences between LR and linear SVM and what is the difference in the effect/concept of their hyper-parameters tuning?

Answers:

Q1: In our model performance is more important to us. Accuracy represnts how much we are "correct". it's calculated by $(TP+TN)/(TP+TN+FP+FN)$. But, some times if our data is imbalanced (almost every one is "sick") and our naive model says everyone is sick we will have high accuracy. which means accuracy is not a good way to asses our model. therefore, we should use other methods to asses our model. A good way to asses our model is using F1 which is: $(2 \text{precisionrecall}) / (\text{Percision} + \text{Sensitivity})$. Percision = $TP / (TP+FP)$ -> total predicted positive Sensitivity= $TP / (TP+FN)$ -> Corecctly predicted positive. With F1 you can weight your model better, it has the highest score if FP and FN have simillar values.

Q2: The first model uses fewer features which means it can have higher correlation between the features and the labels. But! it may cause over-fitting because our dataset is now narrowed to only 2 features. That being said BP and BMI are a good features which can predict heart attack quite well, too much data may cause under-fitting. The second mode. has much more features which might be good, but it can cause under-fitting our model will not "learn" cause we have confusing data which will make it unable to classify or wrongly classifying. In general more data/features is good and better and it can save us from being over-fitting and being more general in our learning.

Q3: The histologist should use non linear SVM. we know that the data he recieved in inseparable (for the human eye at least). Therefore, he can't use any linear method to sepreate the data using any linear line. this is way he should use a nonlinear SVM.

Q4: LR is a classifier that tries to get the maximum likelihood of label to occur. by doing so using probability we get a "smooth" likelihood of prediction to occur using statistical models. LR is influenced by its hyper-parameter (lambda) which is the weight for penalty for incorrect classifying. setting high value for lambda may cause over-fitting, setting a really low lambda value may cause under-fitting. using lambda LR is less influenced by outliers which is one of the "weakness" in LR. linear SVM is a classifier which is trying to maximize the margin between two support vectors. SVM in more general in its method. SVM contains two hyper parameters, (Capacity/C and gamma). C is the penalty weight we want for wrong classification (penalty emphasizes). gamma is how much we want our data to fit. high gamma values may cause over-fitting.

```
In [1]: import pandas as pd
import numpy as np
from pathlib import Path
import random
%load_ext autoreload

# to get the same result each time we run
random.seed(10)
# create a path for the file
filepath = Path.cwd().joinpath('HW2_data.csv')
# load csv file
data = pd.read_csv(filepath)
```

```
In [2]: # check data size
print(f"Data Size: {data.shape}")
#See the first 3 patients to check the data
data.head(3)
```

Data Size: (565, 18)

```
Out[2]:
```

	Age	Gender	Increased Urination	Increased Thirst	Sudden Weight Loss	Weakness	Increased Hunger	Genital Thrush	Visual Blurring	Itching	Irritability	Delayed Healing	Partial Paresis	Muscle Stiffness	Hair Loss
0	45	Male	No	No	No	Yes	No	No	No	Yes	No	No	Yes	No	Yes
1	42	Male	No	No	No	No	No	No	No	No	No	No	No	No	Yes
2	45	Male	Yes	Yes	No	Yes	No	Yes	No	No	No	Yes	No	No	Yes

```
In [3]: # pre-process data
# remove rows without data - "yes"/"no" are not something we should estimate
data = data.dropna()

#convert categorical value to 0 and 1
data = data.replace(['No', 'Yes', 'Negative', 'Positive', 'Female', 'Male'], [0, 1, 0, 1, 0, 1])
data.head(3)
```

```
# check data new size
print(f"Data New Size: {data.shape}")
# extract Y values
Y = data[["Diagnosis"]]
# extract X values
X = data.drop("Diagnosis", axis=1)
# extract Features list
Features = list(X.columns)
```

Data New Size: (523, 18)

```
In [4]: from sklearn.model_selection import train_test_split
# split the data
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2,
                                                    random_state=10, stratify=Y)
```

```
In [5]: import matplotlib.pyplot as plt
import seaborn as sns

percents = pd.DataFrame(np.zeros([17,3]), columns = ['Train %' , 'Test %', 'Delta %'],
                        index=Features, dtype = np.int32)

fig, ax =plt.subplots(4,4, figsize=(20, 14))
fig2, ax2 = plt.subplots(figsize=(12, 3))

for i,feat in enumerate(Features):
    if feat != 'Age':
        sns.countplot(x=feat, hue="Diagnosis", data= data[["Diagnosis", feat]],
                      ax=ax[(i-1)%4][((i-1)//4)])
    if feat == 'Age':
        percents['Train %'][feat] = (np.mean(X_train[feat]))
        percents['Test %'][feat] = (np.mean(X_test[feat]))
        percents['Delta %'][feat] = (np.mean(X_test[feat])) - (np.mean(X_train[feat]))
        percents = percents.rename(index = {feat:"Age (mean)"})
        X[feat][Y.index[(Y == 1)]['Diagnosis'] == True].tolist().hist(
            label = 'Positive', ax = ax2)
        X[feat][Y.index[(Y == 0)]['Diagnosis'] == True].tolist().hist(
            label = 'Negative', ax = ax2)
        ax2.legend(title = 'Diagnosis', fontsize = 'large')
        ax2.set_xlabel(feat, fontsize = 'large')
        ax2.set_ylabel('count', fontsize = 'large')
        #ax.legend(loc='upper right')
    elif feat == 'Gender':
        idx_1 = (X_train == 1).index[(X_train == 1)[feat] == True].tolist()
        percents['Train %'][feat] = (100 * len(idx_1) / len(X_train))
        idx_2 = (X_test == 1).index[(X_test == 1)[feat] == True].tolist()
        percents['Test %'][feat] = (100 * len(idx_2) / len(X_test))
        percents['Delta %'][feat] = (100 * len(idx_2) / len(X_test))-(100 * len(idx_1) / len(X_train))
```

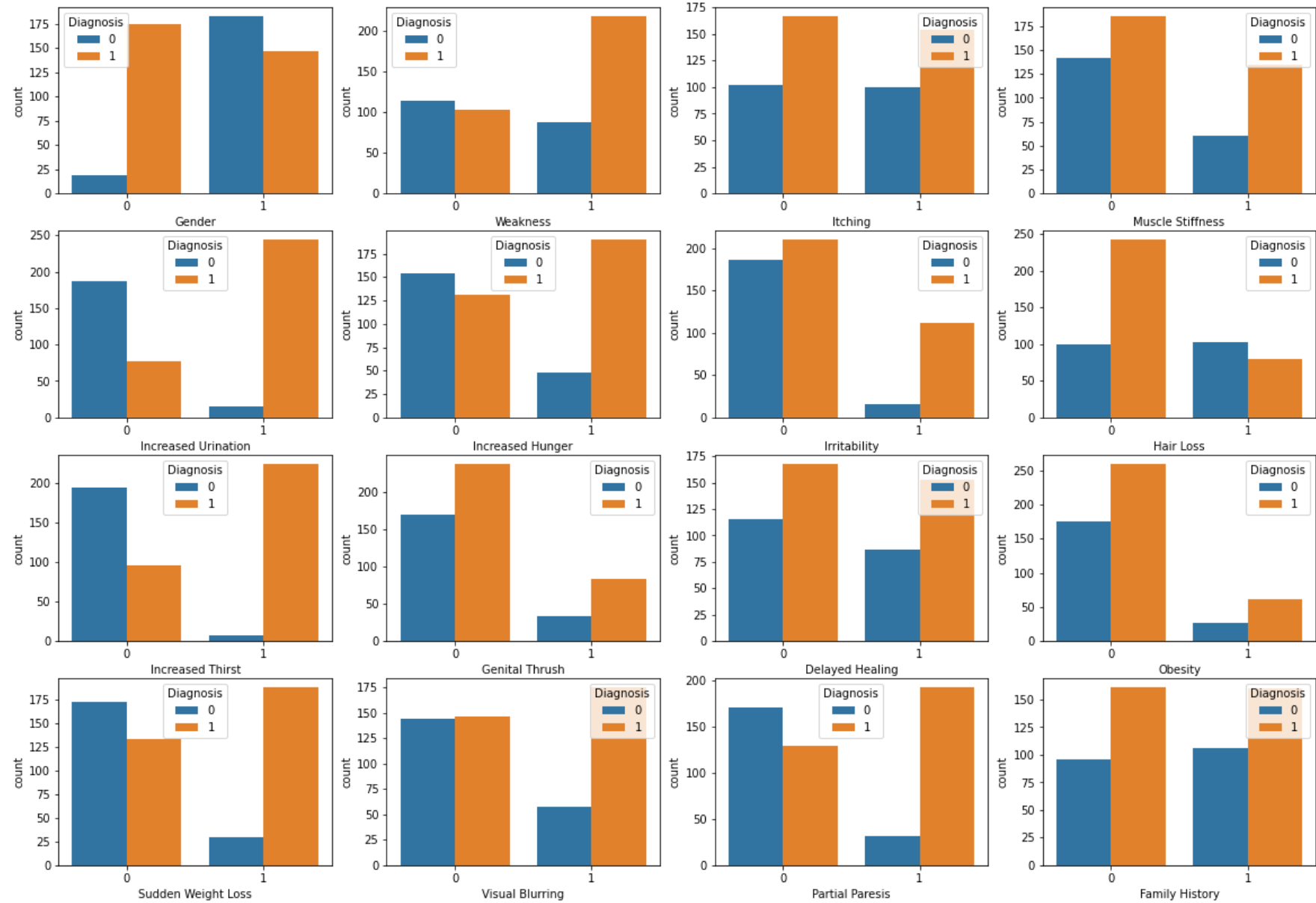
```

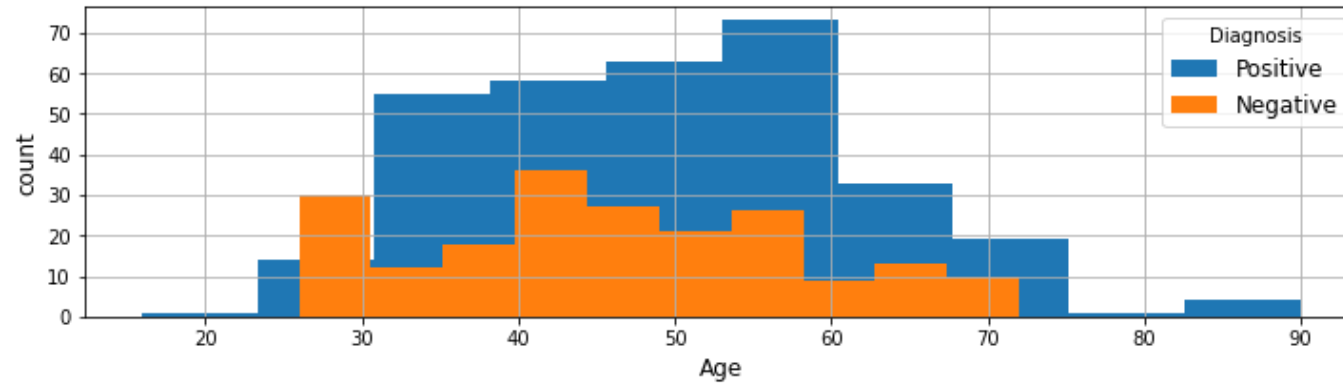
percents = percents.rename(index = {feat:"Gender (Male)"})
elif feat == 'Family History':
    idx_1 = X_train.index[X_train[feat] == True].tolist()
    percents['Train %'][feat] = (100 * len(idx_1) / len(X_train))
    idx_2 = X_test.index[X_test[feat] == True].tolist()
    percents['Test %'][feat] = (100 * len(idx_2) / len(X_test))
    percents['Delta %'][feat] = (100 * len(idx_2) / len(X_test)) - (100 * len(idx_1) / len(X_train))
else:
    idx_1 = (X_train == 1).index[(X_train == 1)[feat] == True].tolist()
    percents['Train %'][feat] = (100 * len(idx_1) / len(X_train))
    idx_2 = (X_test == 1).index[(X_test == 1)[feat] == True].tolist()
    percents['Test %'][feat] = (100 * len(idx_2) / len(X_test))
    percents['Delta %'][feat] = (100 * len(idx_2) / len(X_test)) - (100 * len(idx_1) / len(X_train))

display(percents)

```

	Train %	Test %	Delta %
Age (mean)	48	46	-2
Gender (Male)	61	69	8
Increased Urination	50	45	-4
Increased Thirst	46	38	-8
Sudden Weight Loss	41	40	0
Weakness	58	60	1
Increased Hunger	46	40	-5
Genital Thrush	20	29	9
Visual Blurring	45	39	-6
Itching	49	44	-4
Irritability	24	22	-1
Delayed Healing	47	38	-9
Partial Paresis	43	39	-4
Muscle Stiffness	37	36	-1
Hair Loss	34	33	-1
Obesity	17	15	-1
Family History	49	55	5





3) a.

i.

Q: What issues could an imbalance of features between train and test cause?

A: An imbalance in features between train and test may cause over-fitting and the model will be "bias" toward a specific feature, because there will be "Too much" influence of one feature. it can create a wrong weights prediction and therefore our test results will be poor.

ii.

Q: How could you solve the issue?

A: We can solve this issue by splitting the train and test with the same ratio of features with the "stratify" option. if we can't do that we can use methods like KFold to help us distribute our train and test better for the training.

```
In [6]: # a table that shows cross correlation between 2 features with positive diagnosis

cross_features = pd.DataFrame(np.zeros([16,16]), columns = Features[1:],
                              index=Features[1:], dtype = np.int32)
cm = sns.light_palette("green", as_cmap=True)
ixd = {}
for feat in Features[1:]:
    ixd[feat] = (X[feat]==1)[Y.index[(Y == 1)['Diagnosis'] == True].tolist()]


for feat in Features[1:]:
    for feat2 in Features[1:]:
        cross_features[feat][feat2]=sum(1 for i in Y.index[
            (Y == 1)['Diagnosis']==True] if ixd[feat][i] and ixd[feat2][i])

cross_features = cross_features.style.\
    set_caption("Cross-Correlation between features with positive diagnosis").set_table_styles([
        {'selector': 'caption', 'props': [('color', 'green'),('font-size', '16px')]}]).\

```

```
background_gradient(cmap=cm)
display(cross_features)
```

Cross-Correlation between features with positive diagnosis

	Gender	Increased Urination	Increased Thirst	Sudden Weight Loss	Weakness	Increased Hunger	Genital Thrush	Visual Blurring	Itching	Irritability	Delayed Healing	Partial Paresis	Muscle Stiffness	H Lc
Gender	147	114	100	74	101	75	66	71	67	63	65	70	55	
Increased Urination	114	244	193	161	178	157	63	135	122	86	123	161	105	
Increased Thirst	100	193	225	146	175	146	55	140	122	79	121	153	103	
Sudden Weight Loss	74	161	146	188	148	127	47	100	95	68	98	124	89	
Weakness	101	178	175	148	218	139	50	136	116	80	116	147	106	
Increased Hunger	75	157	146	127	139	190	40	107	100 	78	109	134	93	
Genital Thrush	66	63	55	47	50	40	83	36	51	43	47	28	33	
Visual Blurring	71	135	140	100	136	107	36	175	100	57	85	128	87	
Itching	67	122	122	95	116	100	51	100	154	65	95	100	72	
Irritability	63	86	79	68	80	78	43	57	65	111	63	66	61	
Delayed Healing	65	123	121	98	116	109	47	85	95	63	153	108	77	
Partial Paresis	70	161	153	124	147	134	28	128	100	66	108	192	90	
Muscle Stiffness	55	105	103	89	106	93	33	87	72	61	77	90	135	
Hair Loss	64	57	41	37	46	43	36	38	31	41	34	31	29	
Obesity	31	48	48	42	45	36	20	42	27	24	19	32	40	
Family History	74	120	112	84	108	86	41	89	74	57	73	88	61	

3) d.

i.

Q: Was there anything unexpected?

A: the difference in the gender when the label is negative is a bit unexpected. it seems there are only a few females with "negative" label, which doesn't feel statistically representative.

ii.

Q: Are there any features that you feel will be particularly important to your model? Explain why.

A: We can see that "Increased Urination" and "Increased Thirst" has the biggest difference between their values when the label is "positive" and an opposite effect when the label is "negative", which means that they can be a critical indication for the label.

```
In [7]: #4)
#encoding our data as one hot vector

from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder

one_hot_data = data.drop("Age", axis=1) # Age is not binary
label_encoder = LabelEncoder()
one_hot_encoder = OneHotEncoder(sparse=False)
one_hot_encoded = one_hot_encoder.fit_transform(one_hot_data)

print(one_hot_encoded)
print(one_hot_encoded.shape)

[[0.  1.  1.  ... 0.  1.  0.]
 [0.  1.  1.  ... 0.  1.  0.]
 [0.  1.  0.  ... 1.  1.  0.]
 ...
 [1.  0.  0.  ... 1.  0.  1.]
 [0.  1.  1.  ... 0.  1.  0.]
 [0.  1.  1.  ... 0.  1.  0.]]
(523, 34)
```

5)

a.i

```
In [8]: from sklearn.metrics import confusion_matrix, plot_confusion_matrix, roc_auc_score
import matplotlib.pyplot as plt

calc_TN = lambda y_true, y_pred: confusion_matrix(y_true, y_pred)[0, 0]
calc_FP = lambda y_true, y_pred: confusion_matrix(y_true, y_pred)[0, 1]
calc_FN = lambda y_true, y_pred: confusion_matrix(y_true, y_pred)[1, 0]
calc_TP = lambda y_true, y_pred: confusion_matrix(y_true, y_pred)[1, 1]
```



```

def evaluate_metrics(y_test, y_pred_test):
    TN = calc_TN(y_test, y_pred_test)
    FP = calc_FP(y_test, y_pred_test)
    FN = calc_FN(y_test, y_pred_test)
    TP = calc_TP(y_test, y_pred_test)
    PPV = TP / (TP + FP)
    SE = TP / (TP + FN)
    SP = TN / (TN + FP)
    PPV = TP / (TP + FP)
    ACC = (TP + TN) / (TP + TN + FP + FN)
    F1 = (2 * SE * PPV) / (SE + PPV)
    return SE, SP, PPV, ACC, F1

def plot_print_model_results(model, x_test, y_test):
    y_pred_test = model.predict(x_test)
    y_proba_test = model.predict_proba(x_test)
    plot_confusion_matrix(model, x_test, y_test, cmap=plt.cm.Blues)
    plt.grid(False)
    plt.show()

    [SE, SP, PPV, ACC, F1] = evaluate_metrics(y_test, y_pred_test)
    print('Sensitivity: {:.2f}'.format(SE))
    print('Specificity: {:.2f}'.format(SP))
    print('Precision: {:.2f}'.format(PPV))
    print('Accuracy: {:.2f}'.format(ACC))
    print('F1: {:.2f}'.format(F1))
    print('AUROC: {:.2f}'.format(roc_auc_score(y_test, y_proba_test[:,1])))

def printB(s):
    print('\033[1m'+s+'\033[0m')

```

```

In [9]: #linear SVC
from sklearn.model_selection import StratifiedKFold as SKFold
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline

k_fold_num = 5
skf = SKFold(n_splits=k_fold_num, random_state=10, shuffle=True)
C = np.array([0.001, 0.01, 0.1, 1, 10, 100, 1000])
#gamma = np.array([0.01, 0.1, 1, 10, 100, 1000])
scoring = ['accuracy', 'f1', 'precision', 'recall', 'roc_auc']

svc = SVC(probability=True)
pipe = Pipeline(steps=[('scale', StandardScaler()), ('svm', svc)])

```

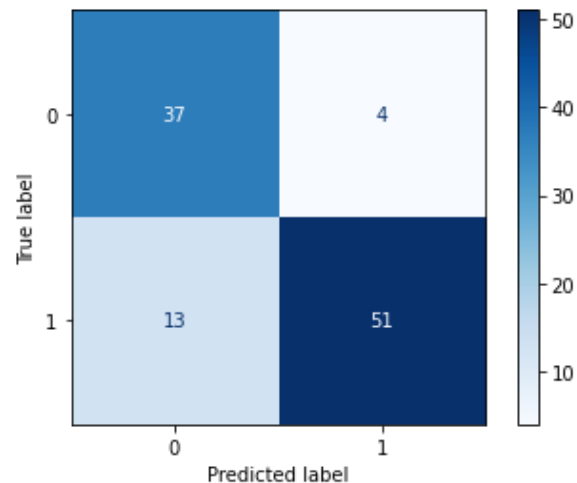
```

params = {'svm_kernel': ['linear'],
          'svm_C': C#,
          #'svm_gamma': gamma,
          }
linear_svm = GridSearchCV(estimator=pipe,
                          param_grid=params,
                          cv=skf,
                          scoring=scoring,
                          refit='roc_auc',
                          verbose=0,
                          return_train_score=True)

linear_svm.fit(X_train, np.ravel(y_train))
best_linear_svm = linear_svm.best_estimator_
print('Linear SVM recieves Best results with: {}'.format(linear_svm.best_params_))
plot_print_model_results(best_linear_svm, X_test, y_test)

```

Linear SVM recieves Best results with: {'svm_C': 0.01, 'svm_kernel': 'linear'}



Sensitivity: 0.80
 Specifcity: 0.90
 Percision: 0.93
 Accuracy: 0.84
 F1: 0.86
 AUROC: 0.96

```

In [10]: #non-linear classifier
params = {'svm_kernel': ['rbf', 'poly'],
          'svm_C': C#,
          #'svm_gamma': gamma,
          }
non_linear_svm = GridSearchCV(estimator=pipe,
                              param_grid=params,

```

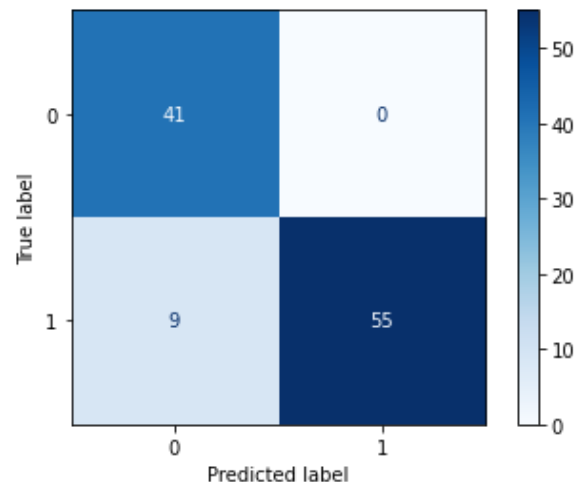
```

cv=skf,
scoring=scoring,
refit='roc_auc',
verbose=0,
return_train_score=True)

non_linear_svm.fit(X_train, np.ravel(y_train))
best_non_linear_svm = non_linear_svm.best_estimator_
print('non-linear SVM recieves Best results with: {}'.format(non_linear_svm.best_params_))
plot_print_model_results(non_linear_svm, X_test, y_test)

```

non-linear SVM recieves Best results with: {'svm__C': 10.0, 'svm__kernel': 'rbf'}



Sensitivity: 0.86
 Specifcity: 1.00
 Percision: 1.00
 Accuracy: 0.91
 F1: 0.92
 AUROC: 0.99

5.c)



we can see that non-linear has better results than linear

6 Feature Selection

```

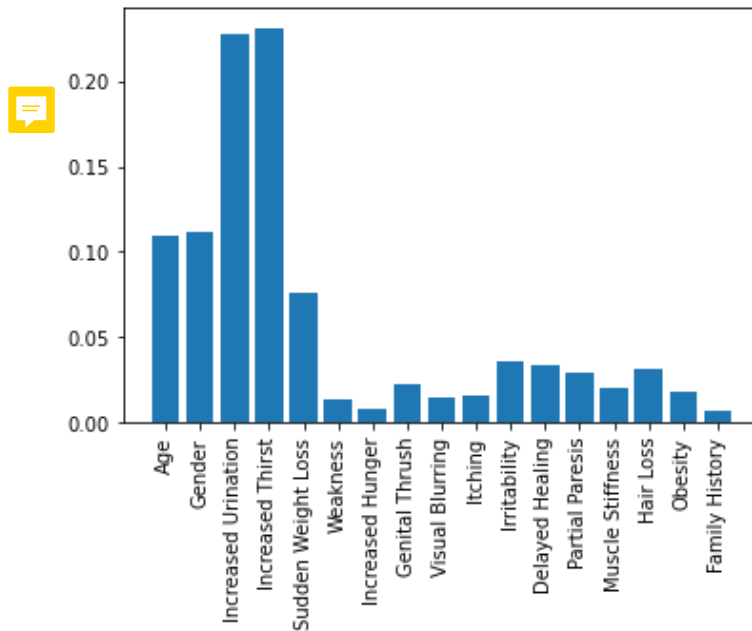
In [11]: from sklearn.ensemble import RandomForestClassifier
random_forest = RandomForestClassifier(n_estimators=10,
                                      max_depth=None, random_state=0,
                                      criterion='gini', max_features='auto')

rfc = random_forest.fit(X_train, np.ravel(y_train))
features_importance = rfc.feature_importances_

```

```
plt.bar(Features, features_importance)
plt.xticks(rotation=90)
plt.show()

print(y_train.shape)
print(np.ravel(y_train).shape)
```



```
(418, 1)
(418,)
```

6.a.i) We can see that Increased Thirst and Increased Urination are the most important feature according to random forest.

6.a.ii) The 2 most important features does match with the exploration we did.

7) Data Separability Visualization

7.a)

```
In [12]: def plt_2d_pca(X_pca,y):
    if isinstance(y, pd.DataFrame):
        y = np.concatenate(y.values.tolist())
    fig = plt.figure(figsize=(8, 8))
    ax = fig.add_subplot(111, aspect='equal')
    ax.scatter(X_pca[y==0, 0], X_pca[y==0, 1], color='b')
    ax.scatter(X_pca[y==1, 0], X_pca[y==1, 1], color='r')
    ax.legend(('Negative', 'Positive'))
```

```

ax.plot([0], [0], "ko")
ax.arrow(0, 0, 0, 1, head_width=0.05, length_includes_head=True, head_length=0.1, fc='k', ec='k')
ax.arrow(0, 0, 1, 0, head_width=0.05, length_includes_head=True, head_length=0.1, fc='k', ec='k')
ax.set_xlabel('$U_1$')
ax.set_ylabel('$U_2$')
ax.set_title('2D PCA')

```

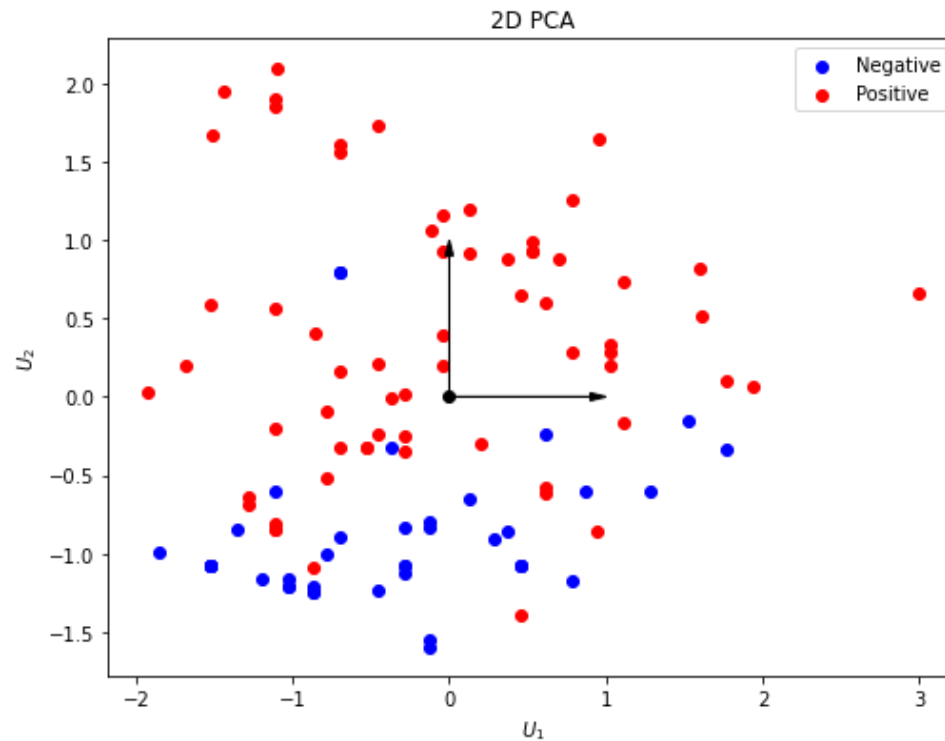
```

In [13]: from sklearn.decomposition import PCA

pca = PCA(n_components=2)
scaler = StandardScaler()
x_train_reduced = pca.fit_transform(X_train)
x_train_reduced = scaler.fit_transform(x_train_reduced)
x_test_reduced = pca.transform(X_test)
x_test_reduced = scaler.transform(x_test_reduced)
#plot figure as requested
print('{:.2f}% of data information was conserved.'.format(100*np.sum(pca.explained_variance_ratio_)))
plt_2d_pca(x_test_reduced,y_test)

```

98.24% of data information was conserved.



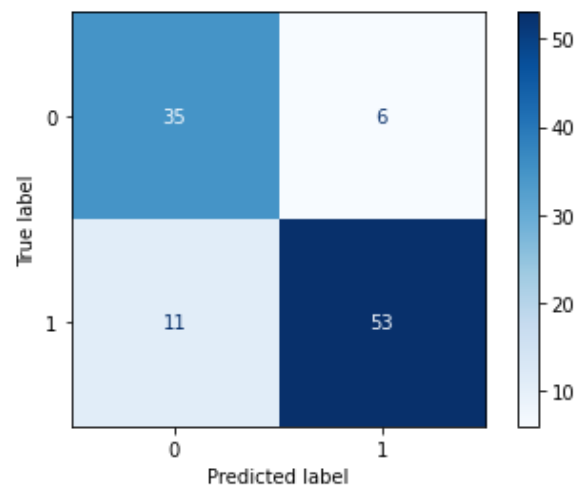
7.b)

7.c)

linear-svm after dimensionality reduced

```
In [14]: linear_svm.fit(x_train_reduced, np.ravel(y_train))
best_linear_svm_reduced = linear_svm.best_estimator_
print('Linear SVM recieves Best results with: {}'.format(linear_svm.best_params_))
plot_print_model_results(best_linear_svm_reduced, x_test_reduced, y_test)
```

Linear SVM recieves Best results with: {'svm__C': 0.1, 'svm__kernel': 'linear'}

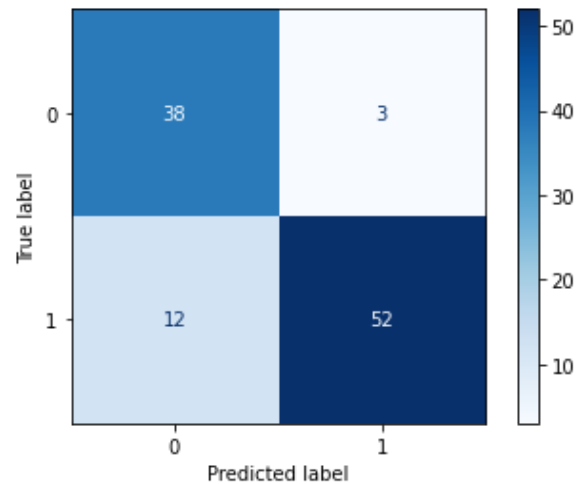


Sensitivity: 0.83
 Specifcity: 0.85
 Percision: 0.90
 Accuracy: 0.84
 F1: 0.86
 AUROC: 0.91

non-linear SVM after dimentionality reduction

```
In [15]: non_linear_svm.fit(x_train_reduced, np.ravel(y_train))
best_non_linear_svm_reduced = non_linear_svm.best_estimator_
print('non-linear SVM recieves Best results with: {}'.format(non_linear_svm.best_params_))
plot_print_model_results(best_non_linear_svm_reduced, x_test_reduced, y_test)
```

non-linear SVM recieves Best results with: {'svm__C': 10.0, 'svm__kernel': 'rbf'}



Sensitivity: 0.81
 Specificity: 0.93
 Percision: 0.95
 Accuracy: 0.86
 F1: 0.87
 AUROC: 0.93

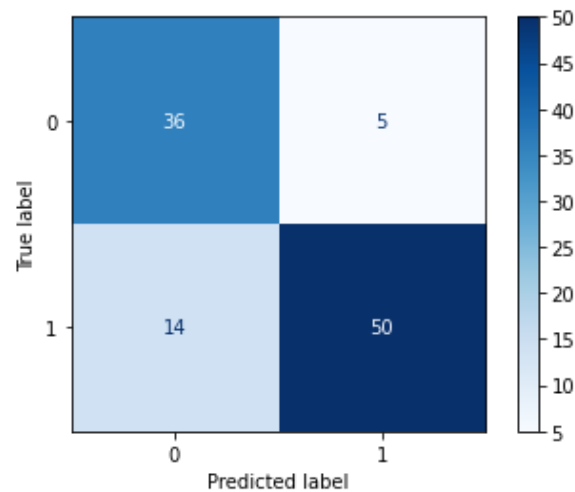
7.d) Training on best 2 features

```
In [16]: x_train_best_features = X_train.iloc[:, [2,3]]
         x_test_best_features = X_test.iloc[:, [2,3]]
```

linear-svm on 2 best features

```
In [17]: linear_svm.fit(x_train_best_features, np.ravel(y_train))
         best_linear_svm_2feat = linear_svm.best_estimator_
         print('Linear SVM recieves Best results with: {}'.format(linear_svm.best_params_))
         plot_print_model_results(best_linear_svm_2feat, x_test_best_features, y_test)
```

Linear SVM recieves Best results with: {'svm__C': 1.0, 'svm__kernel': 'linear'}



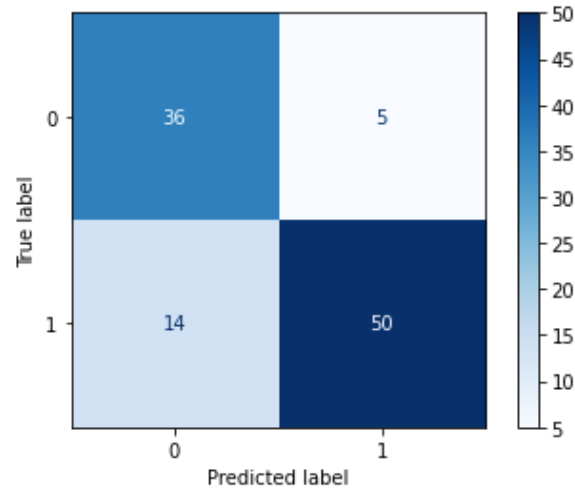
Sensitivity: 0.78
 Specificity: 0.88
 Percision: 0.91
 Accuracy: 0.82
 F1: 0.84
 AUROC: 0.86

non-linear-svm on 2 best features

```

In [18]: non_linear_svm.fit(x_train_best_features, np.ravel(y_train))
          best_non_linear_svm_2feat = non_linear_svm.best_estimator_
          print('non-linear SVM recieves Best results with: {}'.format(non_linear_svm.best_params_))
          plot_print_model_results(best_non_linear_svm_2feat, x_test_best_features, y_test)
  
```

non-linear SVM recieves Best results with: {'svm__C': 1.0, 'svm__kernel': 'poly'}



Sensitivity: 0.78

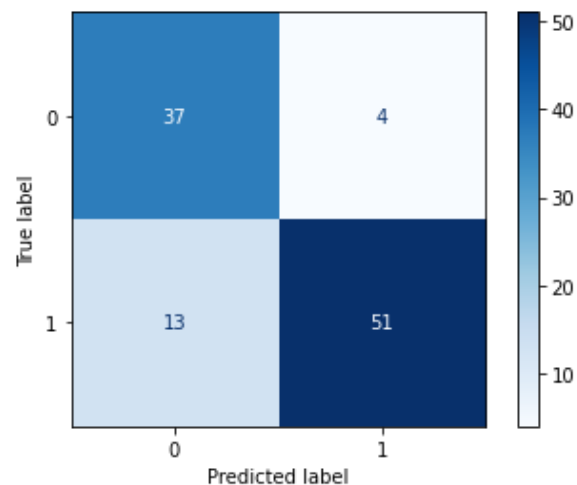
Specifcity: 0.88
 Percision: 0.91
 Accuracy: 0.82
 F1: 0.84
 AUROC: 0.86

```

In [19]: printB('\nSummary:')
          printB('\nLinear SVM')
          plot_print_model_results(best_linear_svm, X_test, y_test)
          printB('\nNon-Linear SVM')
          plot_print_model_results(best_non_linear_svm, X_test, y_test)
          printB('\nLinear SVM + PCA reduced to 2 features')
          plot_print_model_results(best_linear_svm_reduced, x_test_reduced, y_test)
          printB('\nNon-Linear SVM + PCA reduced to 2 features')
          plot_print_model_results(best_non_linear_svm_reduced, x_test_reduced, y_test)
          printB('\nLinear SVM - only 2 best features')
          plot_print_model_results(best_linear_svm_2feat, x_test_best_features, y_test)
          printB('\nNon-Linear SVM - only 2 best features')
          plot_print_model_results(best_non_linear_svm_2feat, x_test_best_features, y_test)
  
```

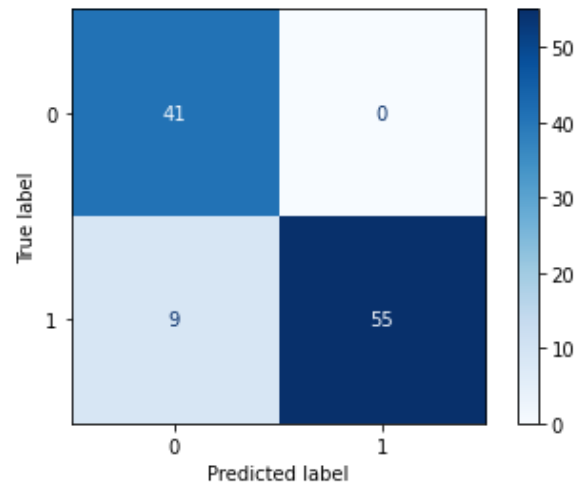
Summary:

Linear SVM



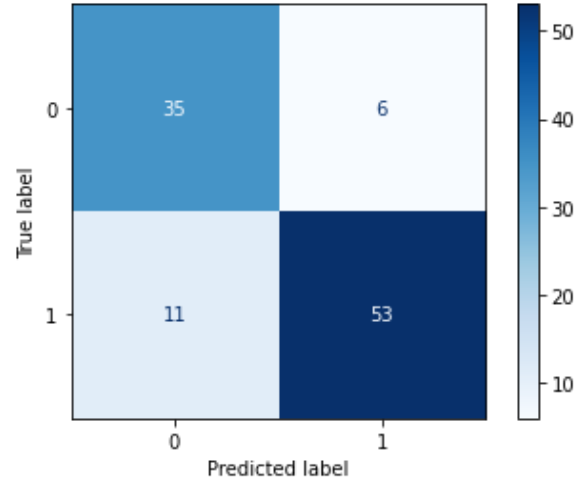
Sensitivity: 0.80
 Specifcity: 0.90
 Percision: 0.93
 Accuracy: 0.84
 F1: 0.86
 AUROC: 0.96

Non-Linear SVM



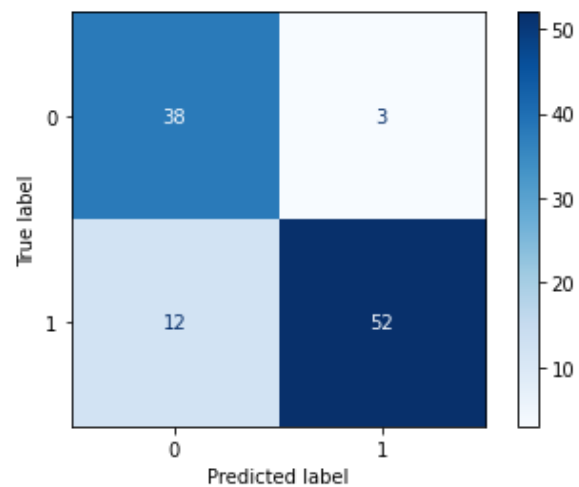
Sensitivity: 0.86
Specificity: 1.00
Precision: 1.00
Accuracy: 0.91
F1: 0.92
AUROC: 0.99

Linear SVM + PCA reduced to 2 features



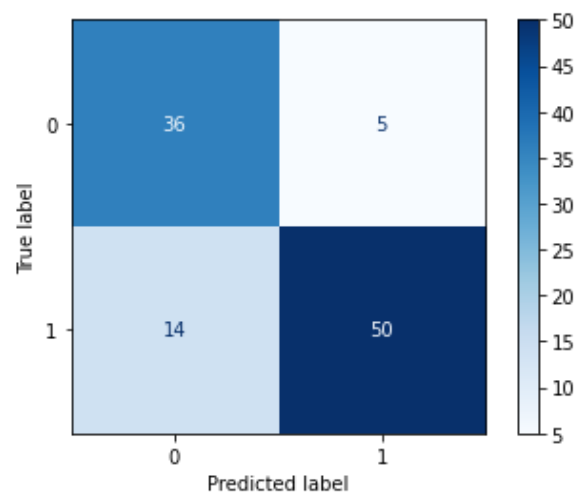
Sensitivity: 0.83
Specificity: 0.85
Precision: 0.90
Accuracy: 0.84
F1: 0.86
AUROC: 0.91

Non-Linear SVM + PCA reduced to 2 features



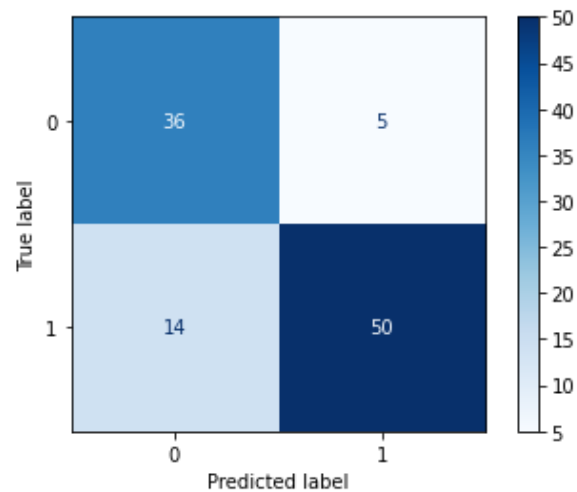
Sensitivity: 0.81
Specificity: 0.93
Precision: 0.95
Accuracy: 0.86
F1: 0.87
AUROC: 0.93

Linear SVM - only 2 best features



Sensitivity: 0.78
Specificity: 0.88
Precision: 0.91
Accuracy: 0.82
F1: 0.84
AUROC: 0.86

Non-Linear SVM - only 2 best features



Sensitivity: 0.78
Specificity: 0.88
Precision: 0.91
Accuracy: 0.82
F1: 0.84
AUROC: 0.86

7.e)

We can see that dimension reduction performed better in contrast to 2 best feature selection.

We could expect that because the dimension reduction gives the optimal 2 orthogonal dimensions, while the selection of the 2 best features are just one possibility out of all "2 orthogonal dimensions" possibilities.

We conclude that with the results from AUROC and F1, which clearly show that the classification of 2 best features alone was not as good as the dimension reduction.

In []: