HW2 - Amit Parizat & Yoel Melul

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.

In our model, the more important evaluation metric is model performance. Like we saw in lecture "*C08 - Practical consideration on training a model II*" in slide 8, accuracy could be high even when the classifier misclassifies most of 'Yes' labeled examples. For example, when 99% percent of the data is labeled 'No' and only 1% is labeled 'Yes' (could be our case since most of the population doesn't have diabetes), even when the classifier labeles 'No' all the examples we obtain high accuracy. In this case, performance statistics such as sensitivity, PPV and F1 would be more informative.

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.

When using only two features we create a simpler model that requires less computational effort to be made and less memory space to accommodate weights or any other information about the model. However, when not taking into consideration all of the data provided, we may miss important information that may have high indicativity on heart attacks. We may also miss trasformations of the neglected features that could yield new valuable features.

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.

The features selected by the histologist probably do not have significant linear correlation with T1D. Thus we do not expect linear models to perform well. We conclude that the best model of the above for such case is the nonlinear SVM model.

Q4

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

Logistic regression is based on probabilistically separating the data through the sigmoid function, asigning each data point to the class with the highest probability of including it. In contrast, linear SVMs are based on a geometrical separation of the data, finding the hyperplane with the widest margin that will separate data points to several classifications. When tuning hyper-parameters in LR we tune a single parameter (λ), while in SVM we tune a set of parameters (C,γ), this is why grid search can be computationally expensive.

Coding Assignment

Part 1

Load the data. Explain any preprocessing.

Import modules

In [1]:
```
import pandas as pd
import numpy as np
from pathlib import Path
import random
import seaborn as sns

import matplotlib.pyplot as plt
import matplotlib as mpl
mpl.style.use(['ggplot'])

from sklearn.model_selection import train_test_split, StratifiedKFold, GridSearchCV
from sklearn.preprocessing import StandardScaler
from IPython.display import display
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import log_loss, hinge_loss, plot_confusion_matrix, confusion_matrix, roc_auc_score, plot_roc_cu
from sklearn.pipeline import Pipeline
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.decomposition import KernelPCA

%load_ext autoreload
random.seed(10)
```

Load data

```
In [2]: file = Path.cwd().joinpath('HW2_data.csv')
        dataset = pd.read_csv(file)
```

Check for nan values

```
In [3]: nan_examples = set()
        nan_features = set()
        for feat in dataset.columns:
            for index in dataset.index:
                if pd.isna(dataset[feat][index]):
                    print(feat, index, end=', ')
                    nan_examples.add(index)
                    nan_features.add(feat)
        print('\n\nThere are %.f examples containing nans.' %len(nan_examples))
        print('There are %.f features containing nans.' %len(nan_features))
```

```
Increased Thirst 521, Increased Thirst 522, Increased Thirst 523, Increased Thirst 524, Increased Thirst 525, Increase
d Thirst 526, Increased Thirst 527, Increased Thirst 528, Increased Thirst 529, Increased Thirst 544, Increased Thirst
545, Increased Thirst 546, Increased Thirst 547, Increased Thirst 548, Increased Thirst 549, Increased Thirst 550, Inc
reased Thirst 551, Increased Thirst 552, Increased Thirst 553, Increased Thirst 554, Sudden Weight Loss 521, Sudden We
ight Loss 525, Sudden Weight Loss 526, Sudden Weight Loss 527, Sudden Weight Loss 528, Sudden Weight Loss 529, Sudden
Weight Loss 530, Sudden Weight Loss 531, Sudden Weight Loss 532, Increased Hunger 530, Increased Hunger 531, Increased
Hunger 532, Increased Hunger 533, Increased Hunger 534, Increased Hunger 535, Increased Hunger 536, Increased Hunger 5
37, Increased Hunger 538, Increased Hunger 539, Increased Hunger 540, Increased Hunger 541, Increased Hunger 542, Geni
tal Thrush 548, Genital Thrush 549, Genital Thrush 550, Genital Thrush 551, Genital Thrush 552, Genital Thrush 553, Ge
nital Thrush 554, Genital Thrush 555, Genital Thrush 556, Genital Thrush 557, Genital Thrush 558, Genital Thrush 559,
Genital Thrush 560, Genital Thrush 561, Itching 520, Itching 530, Itching 531, Itching 532, Itching 533, Itching 534,
Itching 535, Itching 536, Itching 537, Itching 538, Itching 539, Muscle Stiffness 530, Muscle Stiffness 531, Muscle St
iffness 532, Muscle Stiffness 533, Muscle Stiffness 534, Muscle Stiffness 535, Muscle Stiffness 536, Muscle Stiffness
537, Muscle Stiffness 538, Muscle Stiffness 539, Muscle Stiffness 540, Muscle Stiffness 541, Muscle Stiffness 542, Mus
cle Stiffness 543, Muscle Stiffness 544,

There are 42 examples containing nans.
There are 6 features containing nans.
```

Drop examples containing nans

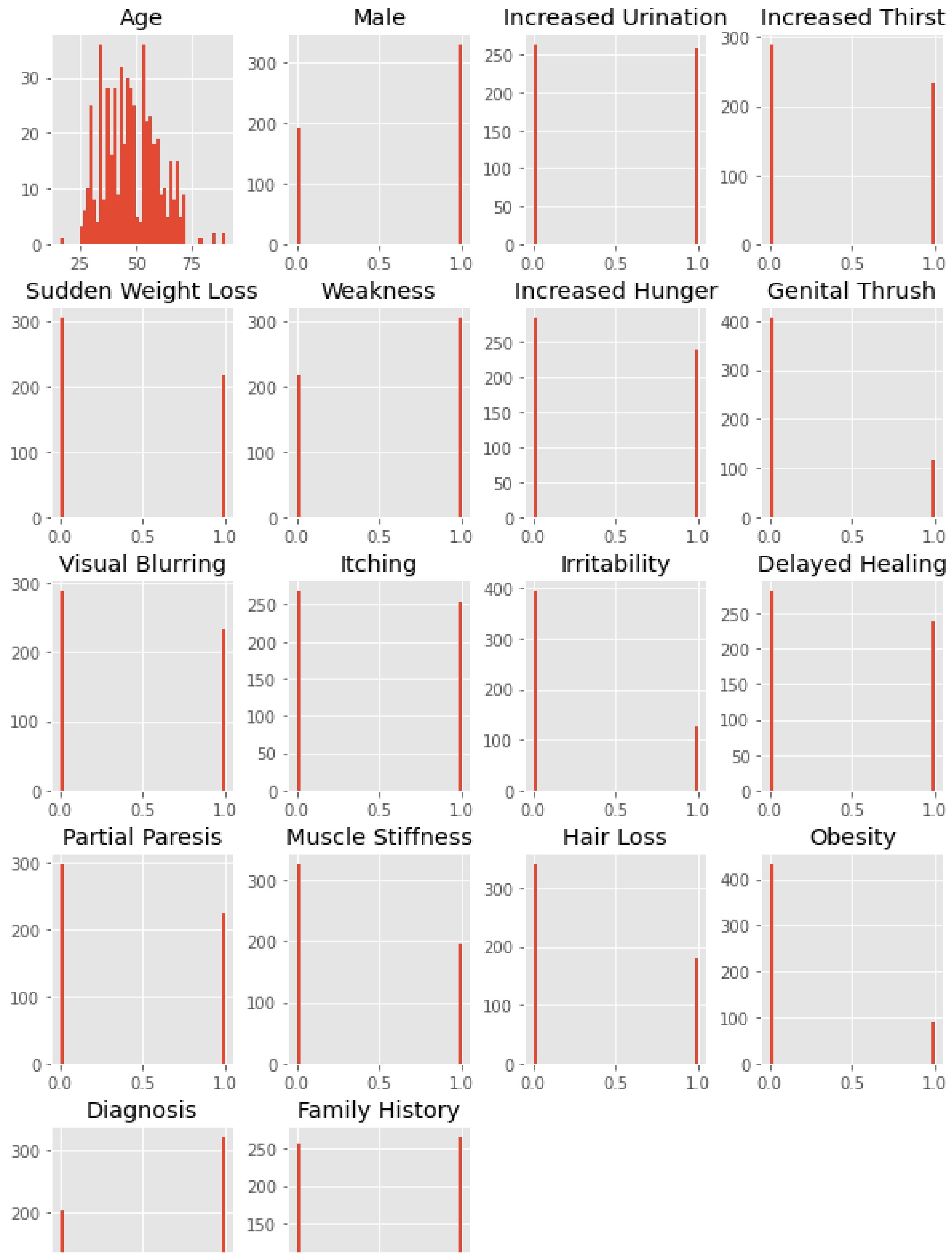
```
In [4]: c_dataset = dataset.drop(nan_examples, axis=0)
```

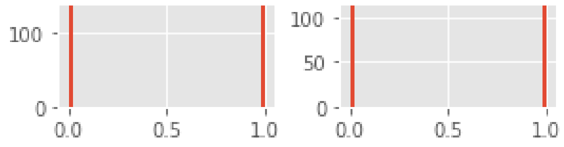
Change 'Yes' to '1' and 'No' to '0'

```
In [5]: c_dataset = c_dataset.rename(columns={'Gender': 'Male'})
        c_dataset = c_dataset.replace(to_replace=['No', 'Negative', 'Female', 'Yes', 'Positive', 'Male'], value=[0, 0, 0, 1,
```

Check the dataset is cleaned

```
In [6]: c_dataset.hist(bins=50, figsize=(10, 15))
        plt.show()
```


Age
Male
Increased Urination
Increased Thirst
Sudden Weight Loss
Weakness
Increased Hunger
Genital Thrush
Visual Blurring
Itching
Irritability
Delayed Healing
Partial Paresis
Muscle Stiffness
Hair Loss
Obesity
Diagnosis
Family History



Split dataset to features and label

In [7]:
```python
feature_list = ['Age', 'Male',
                'Increased Urination', 'Increased Thirst',
                'Sudden Weight Loss', 'Weakness',
                'Increased Hunger', 'Genital Thrush',
                'Visual Blurring', 'Itching', 'Irritability',
                'Delayed Healing', 'Partial Paresis',
                'Muscle Stiffness', 'Hair Loss',
                'Obesity', 'Family History'
               ]
features = c_dataset[feature_list]
label = c_dataset[['Diagnosis']]
```

Part 2

Perform a test-train split of 20% test.

In [8]:
```python
X_train, x_test_orig, Y_train, y_test = train_test_split(features, label, test_size=0.2, random_state=3, stratify=labe
```

Part 3

Provide a detailed visualization and exploration of the data.

a. We show the distribution of the features is similar between test and train:

In [9]:
```python
distribution_feat = X_train.drop(columns='Age').keys()
distribution_train = X_train.drop(columns='Age').mean()*100
distribution_test = x_test_orig.drop(columns='Age').mean()*100
distribution = {'Positive Feature':distribution_feat,
                'Train %':distribution_train,
                'Test %':distribution_test,
                'Delta %':distribution_train-distribution_test}
distribution_table = pd.DataFrame(distribution)

avg_age = {'Train': X_train['Age'].mean(), 'Test': x_test_orig['Age'].mean()}
avg_age['Delta'] = avg_age['Train'] - avg_age['Test']
distribution_age = pd.DataFrame(avg_age, index=['Average Age'])

diagnosis = {'Train %': Y_train['Diagnosis'].mean()*100, 'Test %': y_test['Diagnosis'].mean()*100}
```

```
diagnosis['Delta %'] = diagnosis['Train %'] - diagnosis['Test %']
distribution_diagnosis = pd.DataFrame(diagnosis, index=['Diagnosis'])

display(distribution_diagnosis)
display(distribution_age)
display(distribution_table.style.hide_index())
```

	Train %	Test %	Delta %
Diagnosis	61.483254	60.952381	0.530873

	Train	Test	Delta
Average Age	47.760766	49.095238	-1.334473

Positive Feature	Train %	Test %	Delta %
Male	62.918660	63.809524	-0.890864
Increased Urination	50.000000	47.619048	2.380952
Increased Thirst	45.215311	41.904762	3.310549
Sudden Weight Loss	42.344498	38.095238	4.249260
Weakness	58.133971	60.000000	-1.866029
Increased Hunger	46.411483	41.904762	4.506721
Genital Thrush	21.770335	23.809524	-2.039189
Visual Blurring	43.779904	47.619048	-3.839143
Itching	47.846890	51.428571	-3.581681
Irritability	24.401914	23.809524	0.592390
Delayed Healing	45.693780	46.666667	-0.972887
Partial Paresis	42.583732	43.809524	-1.225792
Muscle Stiffness	37.081340	38.095238	-1.013898
Hair Loss	34.928230	33.333333	1.594896
Obesity	16.028708	20.000000	-3.971292
Family History	50.717703	51.428571	-0.710868

i. What issues could an imbalance of features between train and test cause?

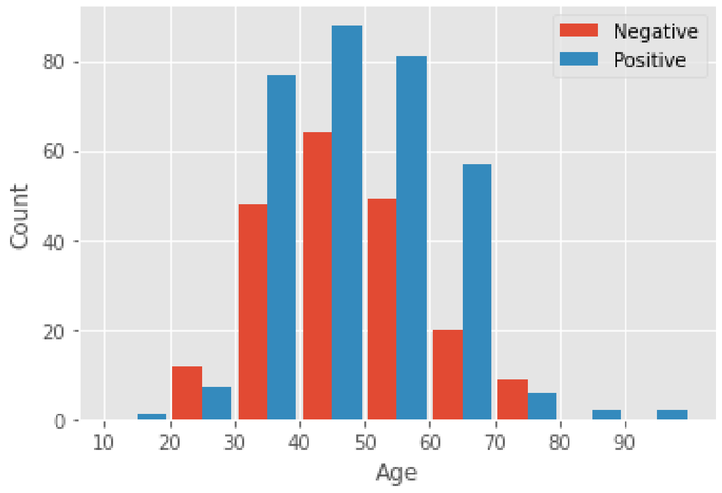
Imbalance of features between train and test sets could cause poor generalization; since the model would train with a set of different distribution from the general population.

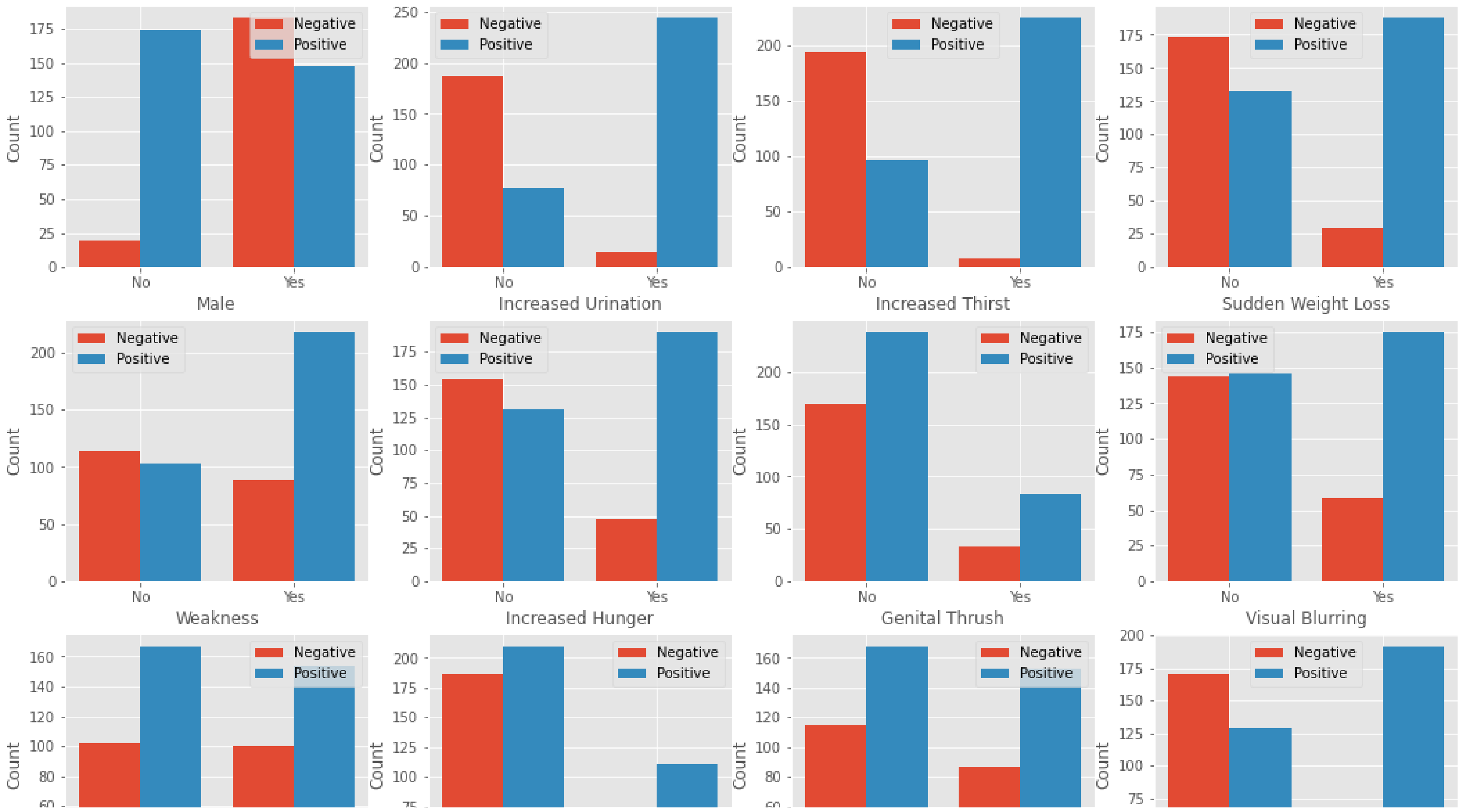
ii. How could you solve the issue?

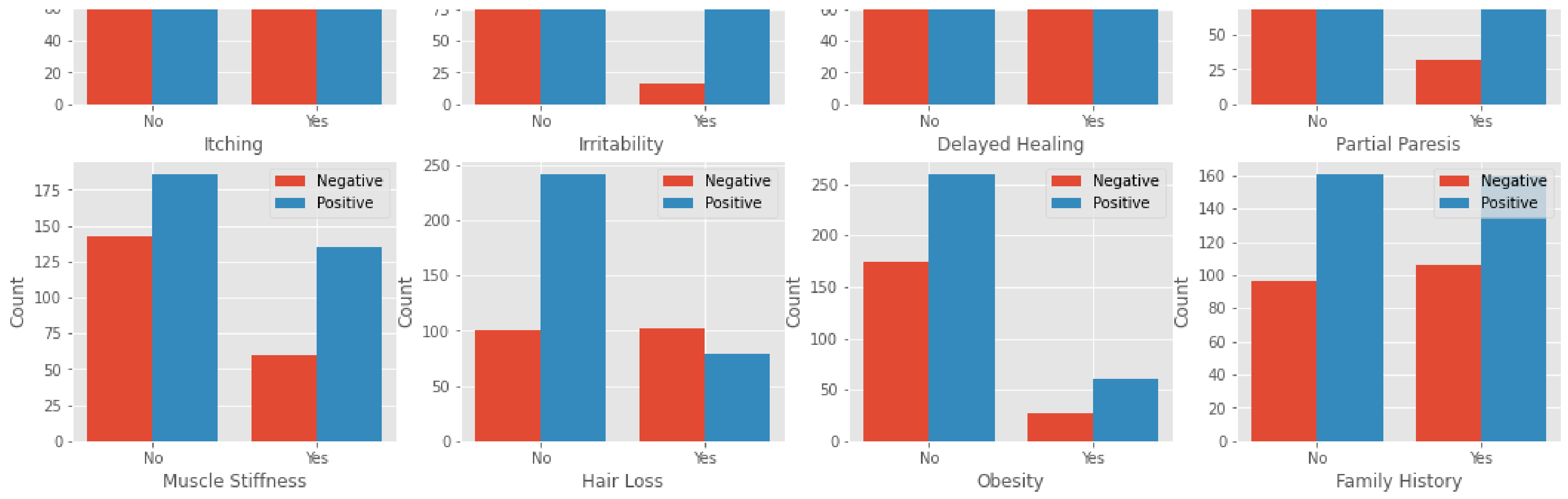
This issue could be solved by stratification, which splits the groups preserving the general distribution of the features for both groups.

b. We show the relationship between feature and label:

In [10]:
```
w = 0.4
w2 = 4.5

age_label0= np.zeros(9)
age_label1= np.zeros(9)

# Plot how many positives/negatives are in each 10 year range
for j in range(9):
    age_label0[j] = sum(features['Age'][i]>=10*(j+1) and features['Age'][i]<(10+10*(j+1)) and label.loc[i]==0 for i in
    age_label1[j] = sum(features['Age'][i]>=10*(j+1) and features['Age'][i]<(10+10*(j+1)) and label.loc[i]==1 for i in

plt.bar([15,25,35,45,55,65,75,85,95], age_label0, w2, label='Negative')
plt.bar([15,25,35,45,55,65,75,85,95] + np.repeat(w2,9), age_label1, w2, label='Positive')
plt.xlabel('Age')
plt.ylabel('Count')
plt.xticks([10,20,30,40,50,60,70,80,90]+ np.repeat(w2/2,9),[10,20,30,40,50,60,70,80,90])
plt.legend()
plt.show()

fig = plt.figure(figsize = [18,16])
for feat in features.drop(columns='Age').columns:
    feat0_label0 = sum([features[feat][i]==0 and label.loc[i]==0 for i in features.index]).item()
    feat0_label1 = sum([features[feat][i]==0 and label.loc[i]==1 for i in features.index]).item()
    feat1_label0 = sum([features[feat][i]==1 and label.loc[i]==0 for i in features.index]).item()
    feat1_label1 = sum([features[feat][i]==1 and label.loc[i]==1 for i in features.index]).item()

    ax = fig.add_subplot(4,4,features.columns.get_loc(feat))
    ax.bar([0, 1], [feat0_label0, feat1_label0], w, label='Negative')
    ax.bar([w, 1+w], [feat0_label1, feat1_label1], w, label='Positive')
    ax.set_xlabel(feat)
    ax.set_ylabel('Count')
    ax.set_xticks([w/2,1+w/2])
    ax.set_xticklabels(['No','Yes'])
    ax.legend()
```

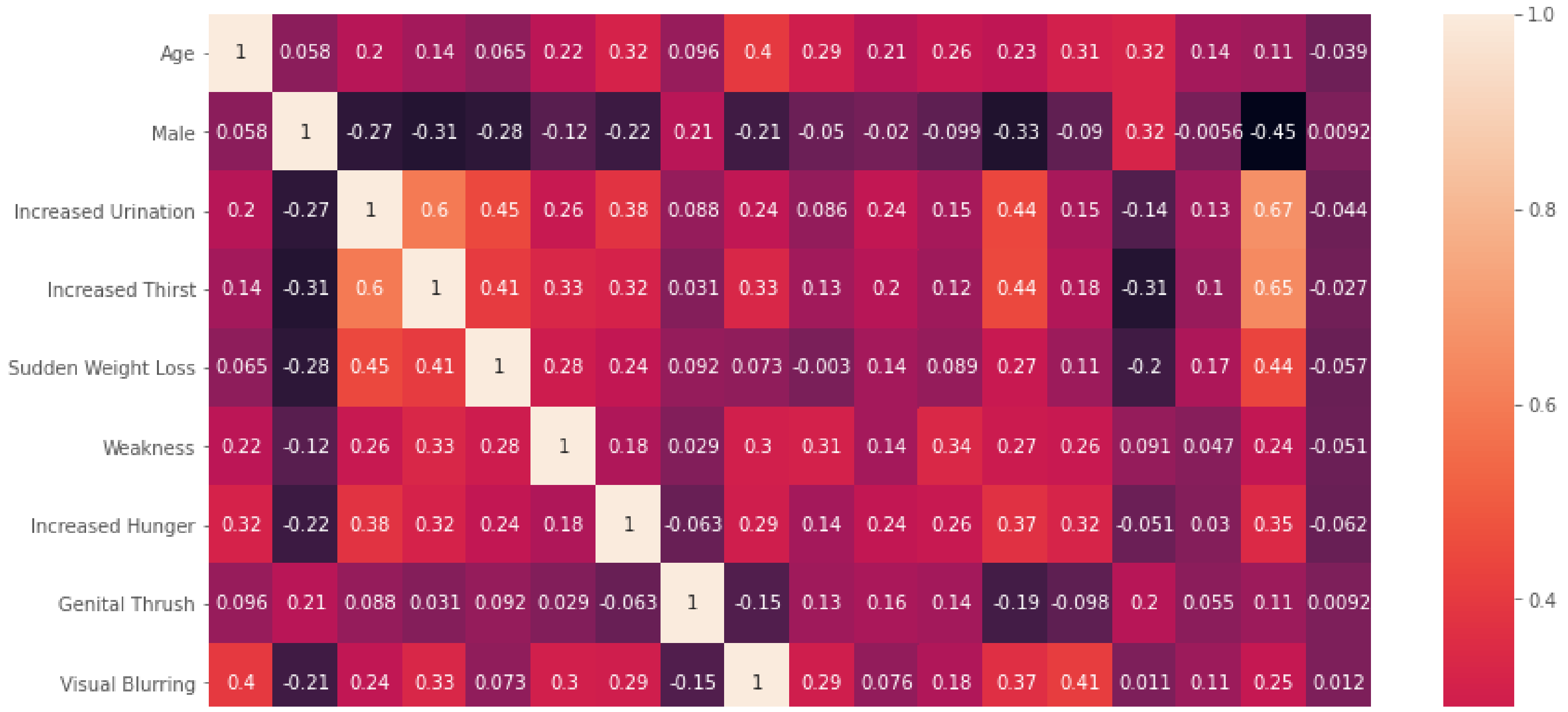

Negative
Positive
Count
0
20
40
60
80
10
20
30
40
50
60
70
80
90
Age

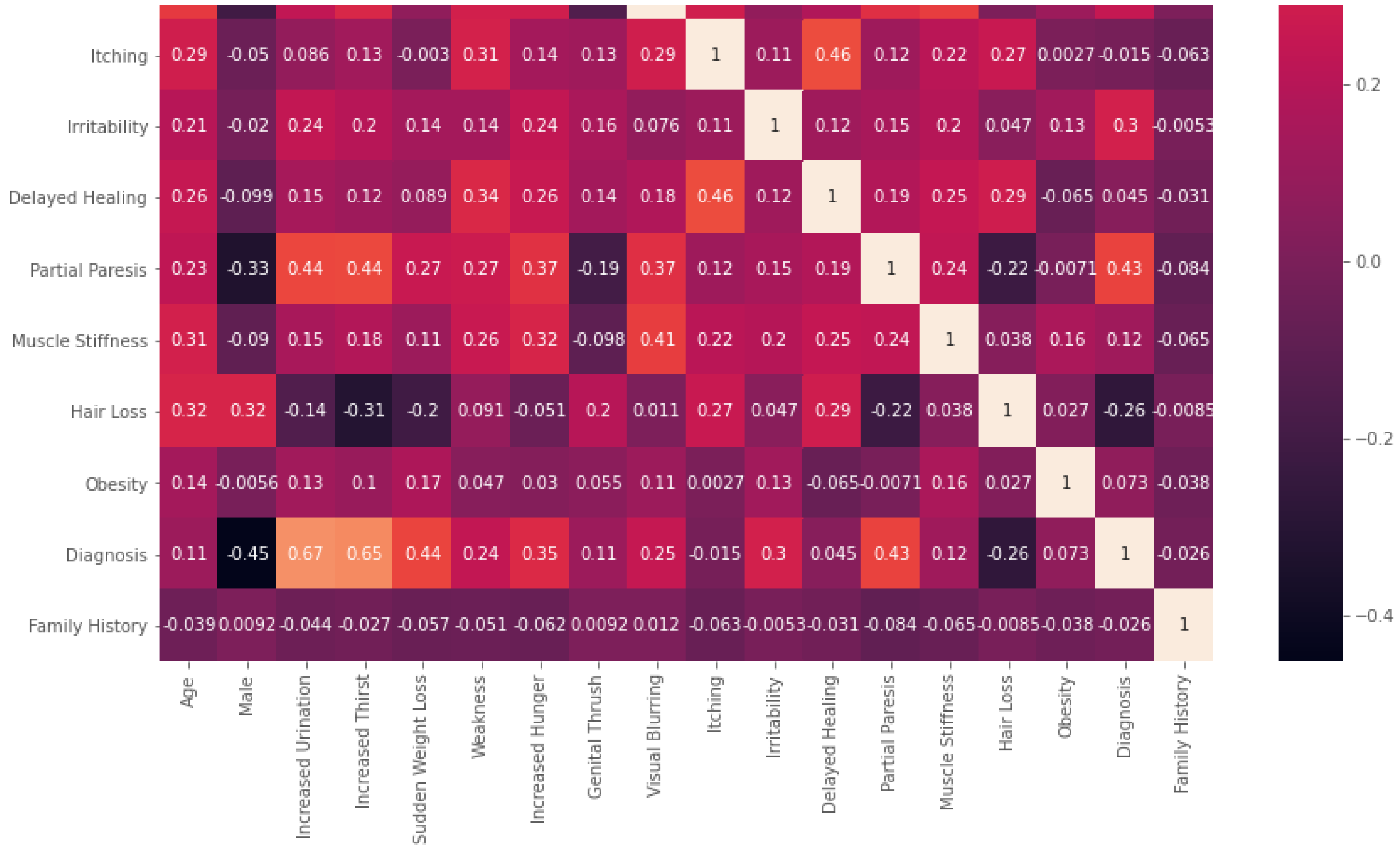

Negative
Positive
Count
No
Yes
Male
Increased Urination
Increased Thirst
Sudden Weight Loss
Weakness
Increased Hunger
Genital Thrush
Visual Blurring



c. Additional plots - Heatmap:

In [11]:
```
plt.figure(figsize = (14,14))
sns.heatmap(c_dataset.corr(), annot = True)
plt.show()
```





	Age	Male	Increased Urination	Increased Thirst	Sudden Weight Loss	Weakness	Increased Hunger	Genital Thrush	Visual Blurring	Itching	Irritability	Delayed Healing	Partial Paresis	Muscle Stiffness	Hair Loss	Obesity	Diagnosis	Family History
Itching	0.29	-0.05	0.086	0.13	-0.003	0.31	0.14	0.13	0.29	1	0.11	0.46	0.12	0.22	0.27	0.0027	-0.015	-0.063
Irritability	0.21	-0.02	0.24	0.2	0.14	0.14	0.24	0.16	0.076	0.11	1	0.12	0.15	0.2	0.047	0.13	0.3	-0.0053
Delayed Healing	0.26	-0.099	0.15	0.12	0.089	0.34	0.26	0.14	0.18	0.46	0.12	1	0.19	0.25	0.29	-0.065	0.045	-0.031
Partial Paresis	0.23	-0.33	0.44	0.44	0.27	0.27	0.37	-0.19	0.37	0.12	0.15	0.19	1	0.24	-0.22	-0.0071	0.43	-0.084
Muscle Stiffness	0.31	-0.09	0.15	0.18	0.11	0.26	0.32	-0.098	0.41	0.22	0.2	0.25	0.24	1	0.038	0.16	0.12	-0.065
Hair Loss	0.32	0.32	-0.14	-0.31	-0.2	0.091	-0.051	0.2	0.011	0.27	0.047	0.29	-0.22	0.038	1	0.027	-0.26	-0.0085
Obesity	0.14	-0.0056	0.13	0.1	0.17	0.047	0.03	0.055	0.11	0.0027	0.13	-0.065	-0.0071	0.16	0.027	1	0.073	-0.038
Diagnosis	0.11	-0.45	0.67	0.65	0.44	0.24	0.35	0.11	0.25	-0.015	0.3	0.045	0.43	0.12	-0.26	0.073	1	-0.026
Family History	-0.039	0.0092	-0.044	-0.027	-0.057	-0.051	-0.062	0.0092	0.012	-0.063	-0.0053	-0.031	-0.084	-0.065	-0.0085	-0.038	-0.026	1

d. Insights:

From the graphs showing the diagnosis according to each feature, we conclude the following:

The following features are indicative of T1D:

- Increased urination
- Increased thirst
- Sudden weight loss
- Increased hunger
- Irritability
- Partial paresis

Other features indicative of T1D but with secondary importance:

- Visual blurring
- Weakness
- Muscle stiffness

We noticed that the number of female subjects that were diagnosed positively was unexpectedly high. We also noticed that most of the subjects with no hair loss were positive, unexpectedly. We concluded that this event might occurr since female subjects tend to have no hair loss. This assumption was confirmed by a positive correlation of 0.32 between being a male and having hair loss, shown in the heatmap.

From the heatmap we observe that the highest positive correlations between feature and diagnosis are increased urination and increased thirst. In addition, sudden weigth loss and partial paresis also show high positive correlation. We also observe the highest negative correlation between diagnosis and being a male and hair loss, which we refered to above.

The features mentioned above with high positive correlation to having T1D are good predictors as expected since they are known symptomes of this disease.

Part 4

Encode all your data as one hot vectors.

This process was done in *Part 1* except for 'Age' feature since there are many values available for this feature. We will only scale this feature:

In [12]:
```
scaler = StandardScaler()
X_train_scaled = X_train.copy()
X_train_scaled['Age'] = scaler.fit_transform(X_train['Age'].values.reshape(-1, 1))
x_test = x_test_orig.copy()
x_test['Age'] = scaler.transform(x_test_orig['Age'].values.reshape(-1, 1))
```

In [13]:
```
X_train_scaled = X_train_scaled.values
x_test = x_test.values
Y_train = Y_train.values
y_test = y_test.values
```

Part 5

Choose, build and optimize Machine Learning Models.

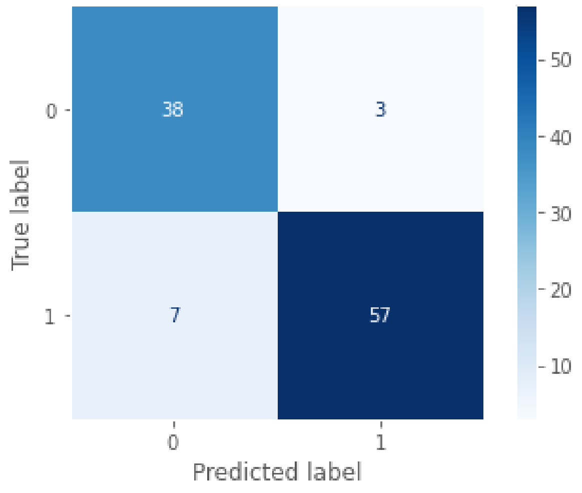
In [14]:
```
calc_TN = lambda y_true, y_pred: confusion_matrix(y_true, y_pred)[0, 0]
calc_FP = lambda y_true, y_pred: confusion_matrix(y_true, y_pred)[0, 1]
calc_FN = lambda y_true, y_pred: confusion_matrix(y_true, y_pred)[1, 0]
calc_TP = lambda y_true, y_pred: confusion_matrix(y_true, y_pred)[1, 1]
```

```
def print_stats(model, x_test, y_test):
    """
    print performance staistics and plot confusion matrix
    inputs: ML model, test set features and models
    """

    display(pd.DataFrame({'Performance statistics': return_stats(model, x_test, y_test)}).transpose())
    plot_confusion_matrix(model, x_test, y_test, cmap=plt.cm.Blues)
    plt.grid(False)

def return_stats(model, x_test, y_test):
    y_pred_test = model.predict(x_test)
    y_pred_proba_test = model.predict_proba(x_test)
    loss = log_loss
    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)
    ACC = (TN + TP) / (TN + FN + TP + FP)
    F1 = 2 * PPV * SE / (PPV + SE)
    AUROC = roc_auc_score(y_test, y_pred_proba_test[:,1])

    if type(model)==LogisticRegression:
        loss = log_loss(y_test, y_pred_proba_test)
    else:
        hinge_y_test = np.array([2*i-1 for i in y_test])
        hinge_y_pred_test = np.array([2*i-1 for i in y_pred_test])
        loss = hinge_loss(hinge_y_test, hinge_y_pred_test)
    return {'Loss': loss, 'Sensitiviy': SE, 'Specificity': SP, 'Accuracy': ACC, 'F1': F1, 'AUROC': AUROC}
```

First, we will train a linear model with no regularization.

```
In [15]: lr = LogisticRegression(random_state=5, max_iter=2000, solver='lbfgs')
         lr.fit(X_train_scaled, Y_train.ravel())
         print_stats(lr, x_test, y_test)
```

	AUROC	Accuracy	F1	Loss	Sensitiviy	Specificity
Performance statistics	0.969893	0.904762	0.919355	0.22375	0.890625	0.926829



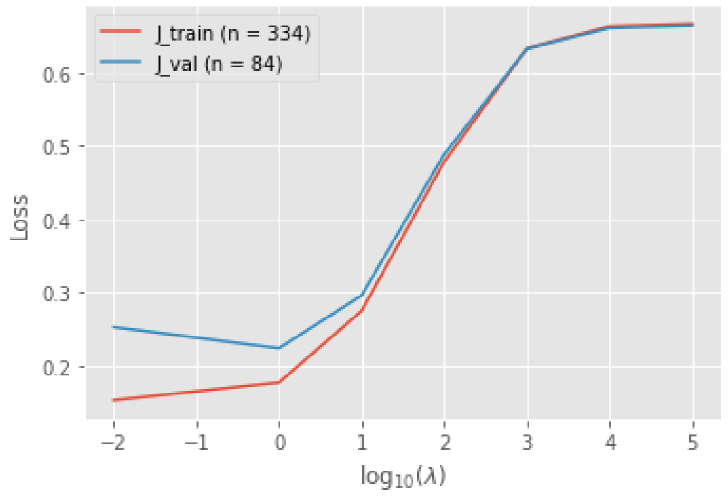
Create validation set and scale it

In [16]:
```
x_train_orig, x_val_orig, y_train, y_val= train_test_split(X_train, Y_train, test_size = 0.2, random_state = 4, strat
```

In [17]:
```
scaler = StandardScaler()
x_train = x_train_orig.copy()
x_val = x_val_orig.copy()
x_train['Age'] = scaler.fit_transform(x_train_orig['Age'].values.reshape(-1, 1))
x_val['Age'] = scaler.transform(x_val_orig['Age'].values.reshape(-1, 1))
```

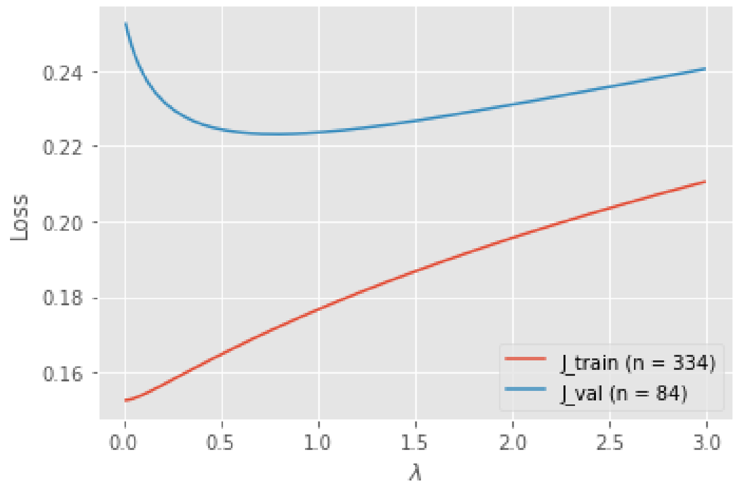
Next we find the best regularization parameters for our linear model.

In [18]:
```
lmbda = np.array([0.01, 1, 10, 100, 1000, 1e04, 1e05])
J_train = np.zeros_like(lmbda)
J_val = np.zeros_like(lmbda)
for idx, lb in enumerate(lmbda):
    log_reg = LogisticRegression(random_state=5, max_iter=2000, solver='lbfgs', C=1/lb)
    log_reg.fit(x_train, y_train.ravel())
    J_train[idx] = log_loss(y_train, log_reg.predict_proba(x_train))
    J_val[idx] = log_loss(y_val, log_reg.predict_proba(x_val))

plt.plot(np.log10(lmbda), J_train)
plt.plot(np.log10(lmbda), J_val)
plt.xlabel('$\log_{10}(\lambda)$')
plt.ylabel('Loss')
plt.legend(['J_train (n = ' + str(x_train.shape[0]) + ')', 'J_val (n = ' + str(x_val.shape[0]) + ')'])
plt.show()
```



We observe that minimum validation loss is for λ ~ 1

In [19]:
```
fine_lmbda = np.arange(0.01, 3, 0.01, dtype=np.float64)
J_train = np.zeros_like(fine_lmbda)
J_val = np.zeros_like(fine_lmbda)
for idx, lb in enumerate(fine_lmbda):
        C = 1/lb
        log_reg = LogisticRegression(random_state=5, max_iter=2000, solver='lbfgs', C=C)
        log_reg.fit(x_train, y_train.ravel())
        J_train[idx] = log_loss(y_train, log_reg.predict_proba(x_train))
        J_val[idx] = log_loss(y_val, log_reg.predict_proba(x_val))

plt.plot(fine_lmbda, J_train)
plt.plot(fine_lmbda, J_val)
plt.xlabel('$\lambda$')
plt.ylabel('Loss')
plt.legend(['J_train (n = ' + str(x_train.shape[0]) + ')', 'J_val (n = ' + str(x_val.shape[0]) + ')'])
plt.show()
```

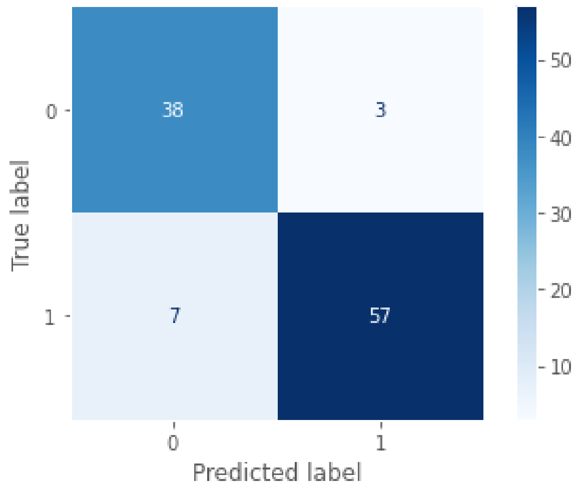


```
In [20]: C = 1/0.5 # best coefficient
```

We train a second linear model, this time with regularization.

```
In [21]: lr_regul = LogisticRegression(random_state=4, max_iter=2000, solver='lbfgs', C=C)
         lr_regul.fit(X_train_scaled, Y_train.ravel())

         print_stats(lr_regul, x_test, y_test)
```

	AUROC	Accuracy	F1	Loss	Sensitiviy	Specificity
Performance statistics	0.971037	0.904762	0.919355	0.213067	0.890625	0.926829



Next we perform 5k cross fold validation with linear, rbf and poly kernels.

```
In [22]: skf = StratifiedKFold(n_splits=5, random_state=10, shuffle=True)
         svc = SVC(probability=True)
         C = np.array([1, 1.5, 2, 2.5, 3])
         pipe = Pipeline(steps=[('svm', svc)])
         svm = GridSearchCV(estimator=pipe, param_grid={'svm__C': C, 'svm__kernel': ['rbf', 'poly', 'linear'], 'svm__degree':
                            scoring=['accuracy','f1','precision','recall','roc_auc'], cv=skf,
                            refit='roc_auc', verbose=1, return_train_score=True)
         svm.fit(X_train_scaled, Y_train.ravel())
         best_svm = svm.best_estimator_
         svm.best_params_
```

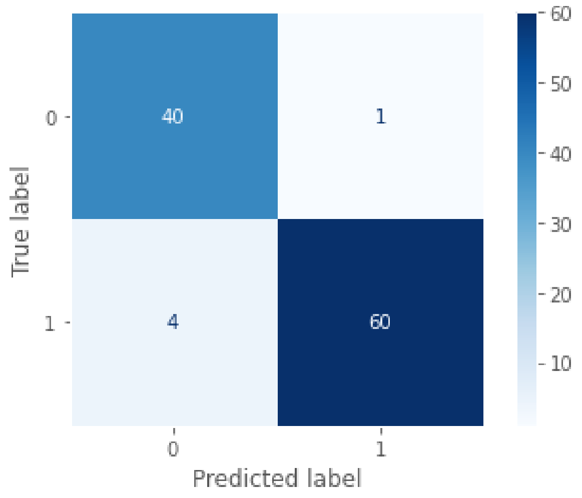
```
Fitting 5 folds for each of 30 candidates, totalling 150 fits
Out[22]: {'svm__C': 2.5, 'svm__degree': 3, 'svm__kernel': 'poly'}
```

5k cross fold validation returned that the best of the model is a polynom of third degree with C = 2.5 .

Next we test the optimized SVM model.

```
In [23]: print_stats(best_svm, x_test, y_test)
```

	AUROC	Accuracy	F1	Loss	Sensitiviy	Specificity
Performance statistics	0.996951	0.952381	0.96	0.095238	0.9375	0.97561

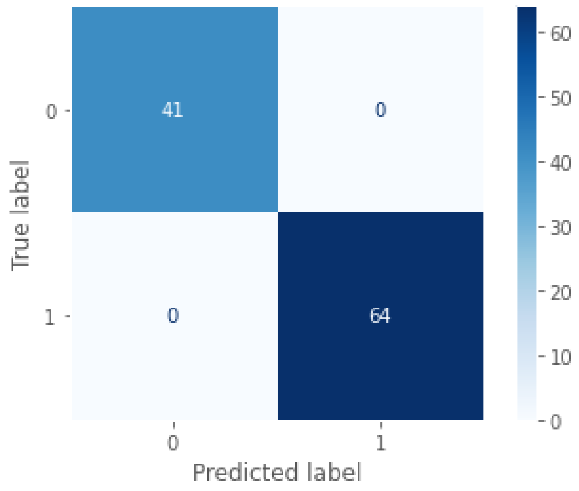


The last non-linear model we train is a Random Forest Classifier.

```
In [24]: rfc = RandomForestClassifier(max_depth=10, random_state=0, criterion='gini')
         rfc.fit(X_train_scaled, Y_train.ravel())
```

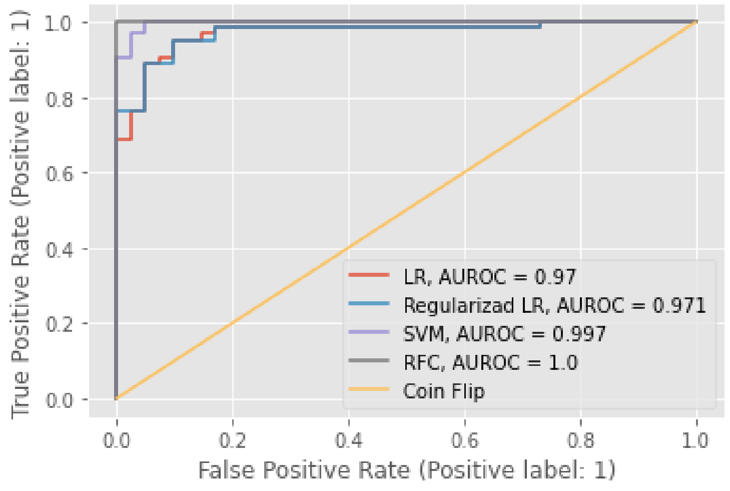
```
print_stats(rfc, x_test, y_test)
```

	AUROC	Accuracy	F1	Loss	Sensitiviy	Specificity
Performance statistics	1.0	1.0	1.0	0.0	1.0	1.0



Finally we plot the ROC of the 4 trained models and show their evaluation metrics.

In [25]:
```
classifiers = [lr, lr_regul, best_svm, rfc]
roc_score = []
plt.figure()
ax = plt.gca()
for clf in classifiers:
    plot_roc_curve(clf, x_test, y_test, ax=ax)
    roc_score.append(np.round_(roc_auc_score(y_test, clf.predict_proba(x_test)[:,1]), decimals=3))
ax.plot(np.linspace(0,1,x_test.shape[0]),np.linspace(0,1,x_test.shape[0]))
plt.legend(('LR, AUROC = '+str(roc_score[0]),'Regularizad LR, AUROC = '+str(roc_score[1]),
            'SVM, AUROC = '+str(roc_score[2]),'RFC, AUROC = '+str(roc_score[3]),'Coin Flip'))
plt.show()
```



In [26]:
```
model_evaluation_metrics = pd.DataFrame({'Linear': return_stats(lr, x_test, y_test),
                                         'Linear Regularized': return_stats(lr_regul, x_test, y_test),
                                         'Polynomial SVM': return_stats(best_svm, x_test, y_test),
                                         'Random Forest Classifier': return_stats(rfc, x_test, y_test)}).transpose()
display(model_evaluation_metrics)
```

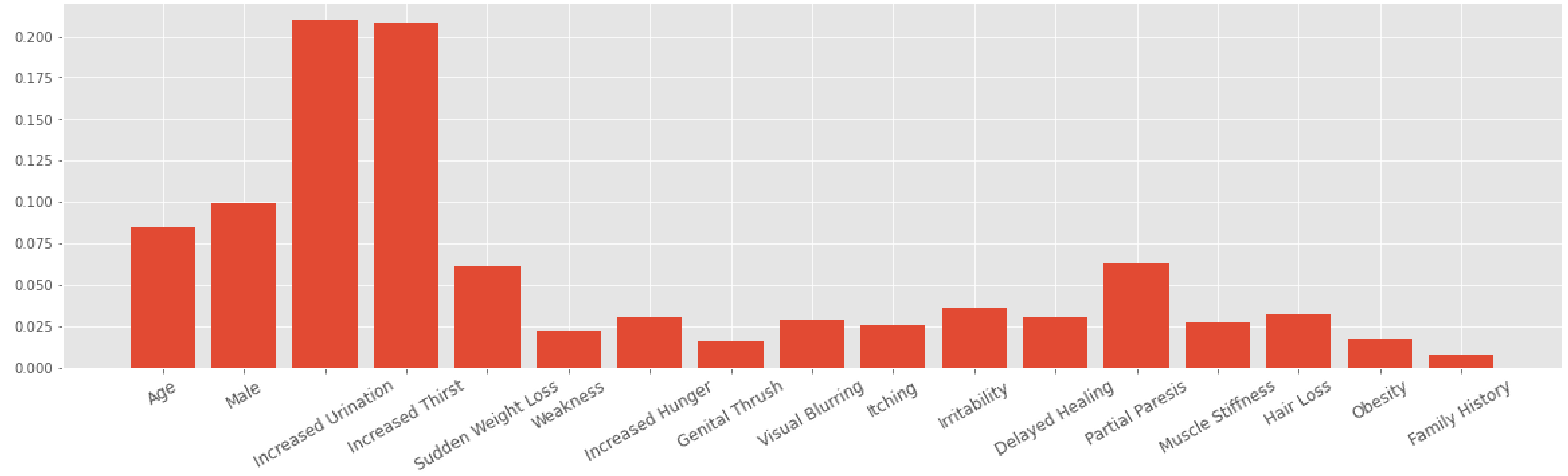
	Loss	Sensitiviy	Specificity	Accuracy	F1	AUROC
Linear	0.223750	0.890625	0.926829	0.904762	0.919355	0.969893
Linear Regularized	0.213067	0.890625	0.926829	0.904762	0.919355	0.971037
Polynomial SVM	0.095238	0.937500	0.975610	0.952381	0.960000	0.996951
Random Forest Classifier	0.000000	1.000000	1.000000	1.000000	1.000000	1.000000

In conclusion, we can observe both from the ROC and the evaluation metrices that non-linear models perform best on this dataset. In particular the random forest classifier model returns a perfect classification with 100% accuracy.

Part 6

Feature Selection

In [27]:
```
feature_importances = rfc.feature_importances_
plt.figure(figsize=(20,5))
plt.bar(feature_list, feature_importances)
plt.xticks(rotation=30, size=12)
plt.show()
```



From the RFC model we can see the 2 most important features are increased urination and increased thirst. This result matches up our prediction from Part 3 that was based on both graphs showing the diagnosis according to each feature and the heatmap.

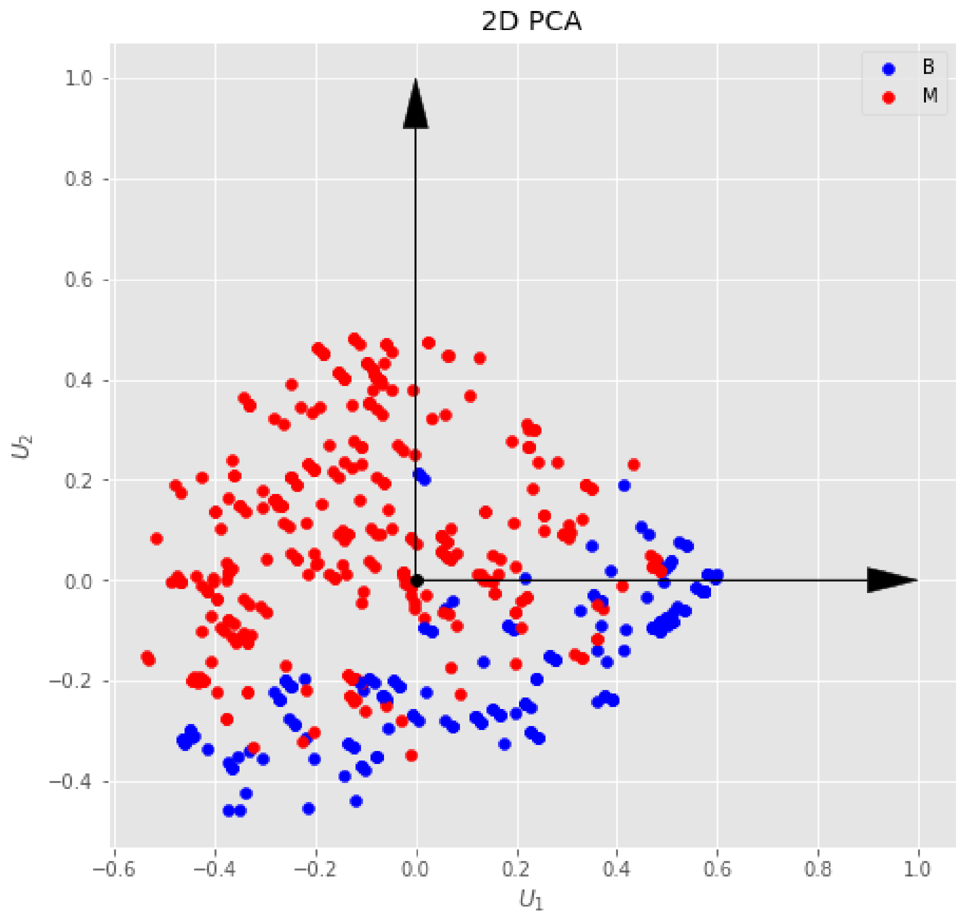
Part 7

Data Separability Visualization

We will reduce the feature dimensionality to 2 using kernel PCA.

```
In [28]: k_pca = KernelPCA(n_components=2, kernel='rbf')
         X_train_pca = k_pca.fit_transform(X_train_scaled)
         x_test_pca = k_pca.transform(x_test)
```

```
In [29]: fig = plt.figure(figsize=(8, 8))
         ax = fig.add_subplot(111, aspect='equal')
         ax.scatter(X_train_pca[Y_train.ravel()==0, 0], X_train_pca[Y_train.ravel()==0, 1], color='b')
         ax.scatter(X_train_pca[Y_train.ravel()==1, 0], X_train_pca[Y_train.ravel()==1, 1], color='r')
         ax.scatter(x_test_pca[y_test.ravel()==0, 0], x_test_pca[y_test.ravel()==0, 1], color='b')
         ax.scatter(x_test_pca[y_test.ravel()==1, 0], x_test_pca[y_test.ravel()==1, 1], color='r')
         ax.legend(('B','M'))
         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')
         plt.show()
```



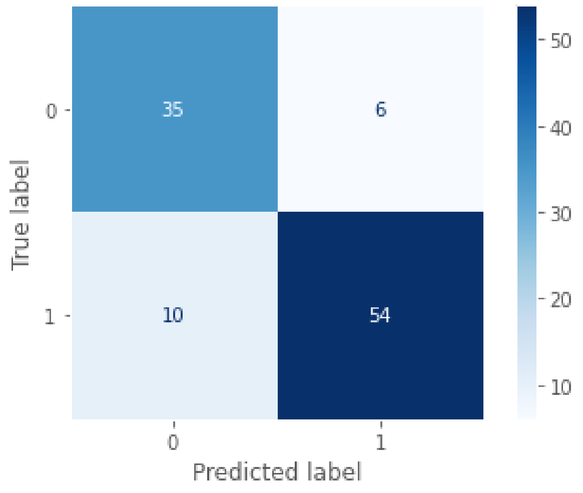
As we can see from the graph above, the reduced data is separable, yet ideal results could not be obtained neither linearly nor non-linearly.

We train the previous models on the dimensionality-reduced training set.

Unregularized logistic regression:

In [30]:
```
lr_pca = LogisticRegression(random_state=5, max_iter=2000, solver='lbfgs')
lr_pca.fit(X_train_pca, Y_train.ravel())
print_stats(lr_pca, x_test_pca, y_test)
```

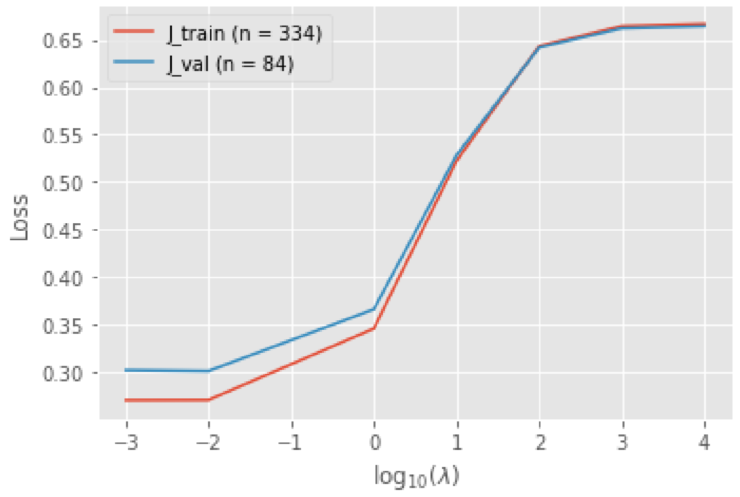
	AUROC	Accuracy	F1	Loss	Sensitiviy	Specificity
Performance statistics	0.929878	0.847619	0.870968	0.369033	0.84375	0.853659



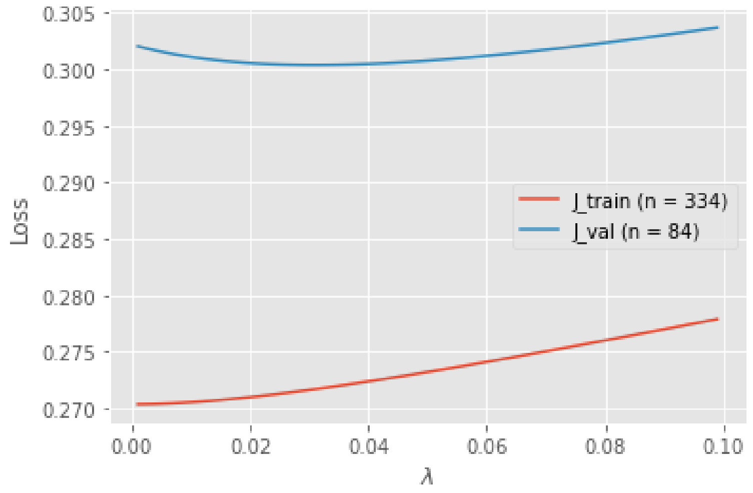
Regularized logistic regression:

In [31]:
```
x_train_pca, x_val_pca, y_train_pca, y_val_pca = train_test_split(X_train_pca, Y_train, test_size = 0.2, random_state
```

In [32]:
```
lmbda = np.array([0.001, 0.01, 1, 10, 100, 1000, 1e04])
J_train = np.zeros_like(lmbda)
J_val = np.zeros_like(lmbda)
for idx, lb in enumerate(lmbda):
    log_reg = LogisticRegression(random_state=5, max_iter=2000, solver='lbfgs', C=1/lb)
    log_reg.fit(x_train_pca, y_train_pca.ravel())
    J_train[idx] = log_loss(y_train_pca, log_reg.predict_proba(x_train_pca))
    J_val[idx] = log_loss(y_val_pca, log_reg.predict_proba(x_val_pca))

plt.plot(np.log10(lmbda), J_train)
plt.plot(np.log10(lmbda), J_val)
plt.xlabel('$\log_{10}(\lambda)$')
plt.ylabel('Loss')
plt.legend(['J_train (n = ' + str(x_train_pca.shape[0]) + ')', 'J_val (n = ' + str(x_val_pca.shape[0]) + ')'])
plt.show()
```

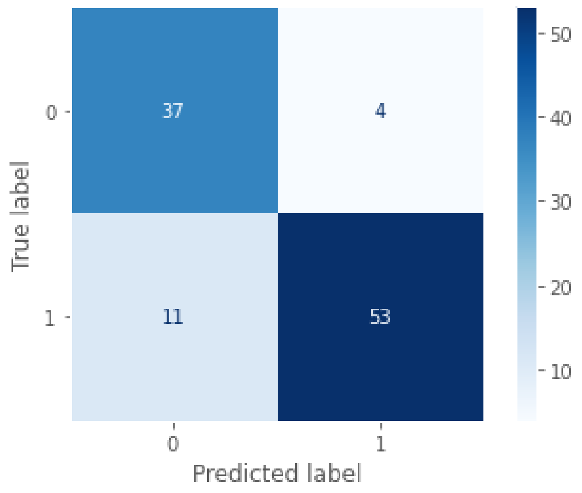


```
In [33]: fine_lmbda = np.arange(0.001, 0.1, 0.001, dtype=np.float64)
         J_train = np.zeros_like(fine_lmbda)
         J_val = np.zeros_like(fine_lmbda)
         for idx, lb in enumerate(fine_lmbda):
                 C = 1/lb
                 log_reg = LogisticRegression(random_state=5, max_iter=2000, solver='lbfgs', C=C)
                 log_reg.fit(x_train_pca, y_train_pca.ravel())
                 J_train[idx] = log_loss(y_train_pca, log_reg.predict_proba(x_train_pca))
                 J_val[idx] = log_loss(y_val_pca, log_reg.predict_proba(x_val_pca))

         plt.plot(fine_lmbda, J_train)
         plt.plot(fine_lmbda, J_val)
         plt.xlabel('$\lambda$')
         plt.ylabel('Loss')
         plt.legend(['J_train (n = ' + str(x_train_pca.shape[0]) + ')', 'J_val (n = ' + str(x_val_pca.shape[0]) + ')'])
         plt.show()
```



In [34]:
```
C = 1/0.03 # Best coefficient
lr_regul_pca = LogisticRegression(random_state=6, max_iter=2000, solver='lbfgs', C=C)
lr_regul_pca.fit(X_train_pca, Y_train.ravel())
print_stats(lr_regul_pca, x_test_pca, y_test)
```

	AUROC	Accuracy	F1	Loss	Sensitiviy	Specificity
Performance statistics	0.932165	0.857143	0.876033	0.329696	0.828125	0.902439



Best SVM:

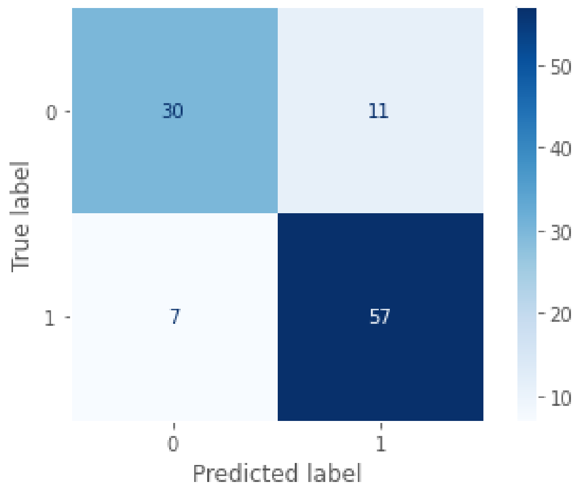
In [35]:
```
C = np.array([0.5,1,2,3])
svm_pca = GridSearchCV(estimator=pipe, param_grid={'svm__C': C, 'svm__kernel': ['rbf', 'poly', 'linear'], 'svm__degree
                      scoring=['accuracy','f1','precision','recall','roc_auc'], cv=skf,
                      refit='roc_auc', verbose=1, return_train_score=True)
svm_pca.fit(X_train_pca, Y_train.ravel())
best_svm_pca = svm_pca.best_estimator_
svm_pca.best_params_
```

```
Fitting 5 folds for each of 24 candidates, totalling 120 fits
```

Out[35]:
```
{'svm__C': 1.0, 'svm__degree': 3, 'svm__kernel': 'poly'}
```

In [36]:
```
print_stats(best_svm_pca, x_test_pca, y_test)
```

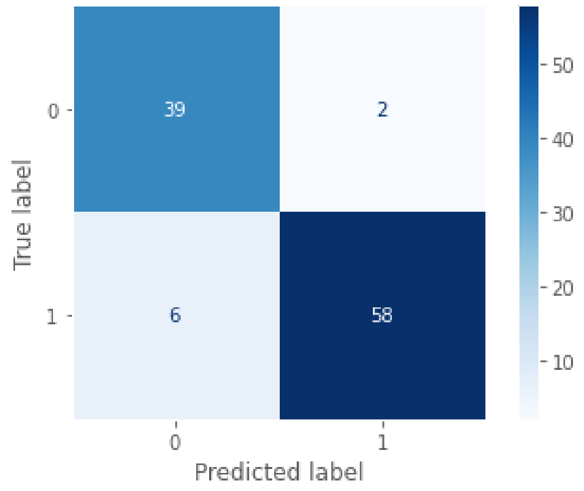
	AUROC	Accuracy	F1	Loss	Sensitiviy	Specificity
Performance statistics	0.924162	0.828571	0.863636	0.342857	0.890625	0.731707



Random Forest Classifier:

In [37]:
```
rfc_pca = RandomForestClassifier(max_depth=10, random_state=0, criterion='gini')
rfc_pca.fit(X_train_pca, Y_train.ravel())
print_stats(rfc_pca, x_test_pca, y_test)
```

	AUROC	Accuracy	F1	Loss	Sensitiviy	Specificity
Performance statistics	0.969512	0.92381	0.935484	0.152381	0.90625	0.95122



We train the previous models on dataset that includes only the two most indicative features.

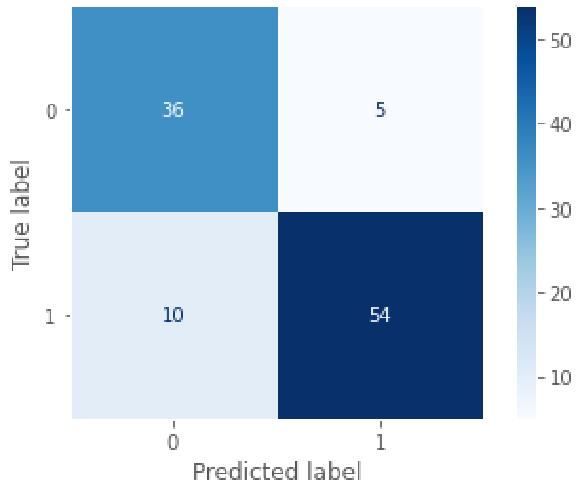
We create the reduced dataset containing only the features: Increased urination, Increased thirst. As we saw in part 6, and also in part 3, these features are the most indicative.

In [38]:
```
X_train_2feat = X_train_scaled[:,[2,3]]
x_test_2feat = x_test[:,[2,3]]
```

Unregularized logistic regression

In [39]:
```
lr_2feat = LogisticRegression(random_state=5, max_iter=2000, solver='lbfgs')
lr_2feat.fit(X_train_2feat, Y_train.ravel())
print_stats(lr_2feat, x_test_2feat, y_test)
```

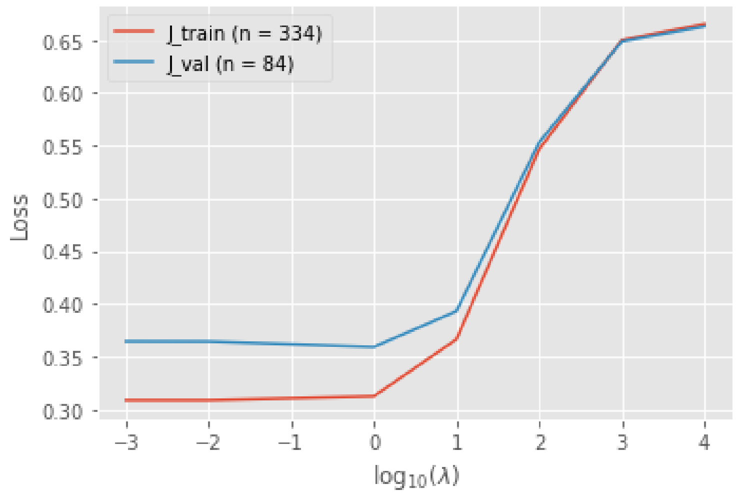
	AUROC	Accuracy	F1	Loss	Sensitiviy	Specificity
Performance statistics	0.893674	0.857143	0.878049	0.353078	0.84375	0.878049



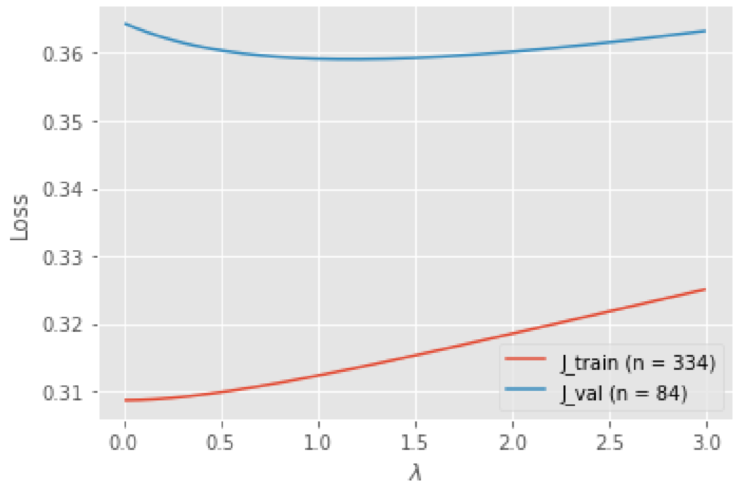
Regularized logistic regression:

In [40]:
```
x_train_2feat, x_val_2feat, y_train_2feat, y_val_2feat = train_test_split(X_train_2feat, Y_train, test_size = 0.2, ra
```

In [41]:
```
lmbda = np.array([0.001, 0.01, 1, 10, 100, 1000, 1e04])
J_train = np.zeros_like(lmbda)
J_val = np.zeros_like(lmbda)
for idx, lb in enumerate(lmbda):
    log_reg = LogisticRegression(random_state=5, max_iter=2000, solver='lbfgs', C=1/lb)
    log_reg.fit(x_train_2feat, y_train_2feat.ravel())
    J_train[idx] = log_loss(y_train_2feat, log_reg.predict_proba(x_train_2feat))
    J_val[idx] = log_loss(y_val_2feat, log_reg.predict_proba(x_val_2feat))

plt.plot(np.log10(lmbda), J_train)
plt.plot(np.log10(lmbda), J_val)
plt.xlabel('$\log_{10}(\lambda)$')
plt.ylabel('Loss')
plt.legend(['J_train (n = ' + str(x_train_2feat.shape[0]) + ')', 'J_val (n = ' + str(x_val_2feat.shape[0]) + ')'])
plt.show()
```

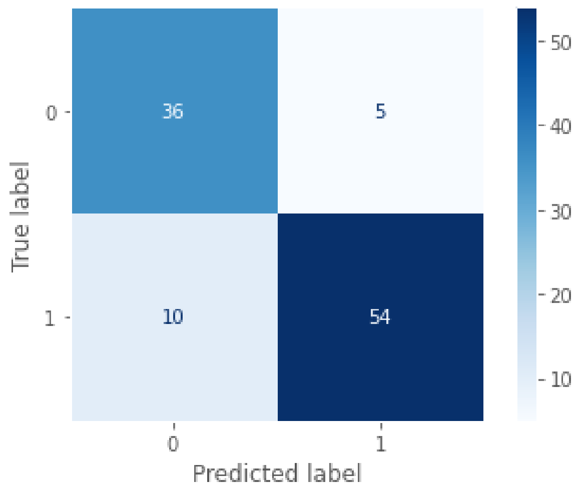


In [42]:
```
fine_lmbda = np.arange(0.01, 3, 0.01, dtype=np.float64)
J_train = np.zeros_like(fine_lmbda)
J_val = np.zeros_like(fine_lmbda)
for idx, lb in enumerate(fine_lmbda):
        C = 1/lb
        log_reg = LogisticRegression(random_state=5, max_iter=2000, solver='lbfgs', C=C)
        log_reg.fit(x_train_2feat, y_train_2feat.ravel())
        J_train[idx] = log_loss(y_train_2feat, log_reg.predict_proba(x_train_2feat))
        J_val[idx] = log_loss(y_val_2feat, log_reg.predict_proba(x_val_2feat))

plt.plot(fine_lmbda, J_train)
plt.plot(fine_lmbda, J_val)
plt.xlabel('$\lambda$')
plt.ylabel('Loss')
plt.legend(['J_train (n = ' + str(x_train_2feat.shape[0]) + ')', 'J_val (n = ' + str(x_val_2feat.shape[0]) + ')'])
plt.show()
```



```
In [43]: C = 1/2 # Best coefficient
         lr_regul_2feat = LogisticRegression(random_state=6, max_iter=2000, solver='lbfgs', C=C)
         lr_regul_2feat.fit(X_train_2feat, Y_train.ravel())
         print_stats(lr_regul_2feat, x_test_2feat, y_test)
```

	AUROC	Accuracy	F1	Loss	Sensitiviy	Specificity
Performance statistics	0.893674	0.857143	0.878049	0.356137	0.84375	0.878049



Best SVM:

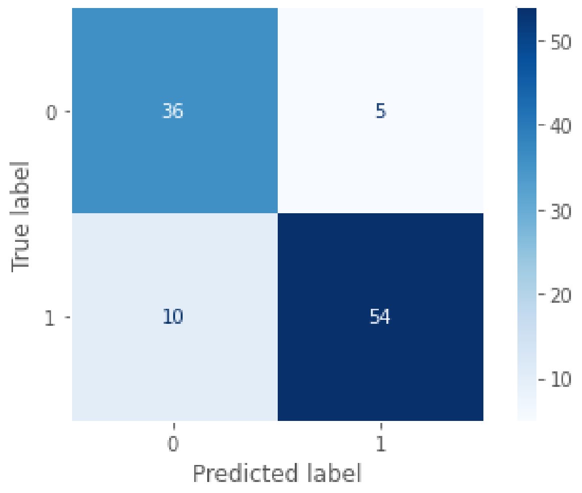
```
In [44]: C = np.array([0.1,0.3,0.5,1])
         svm_2feat = GridSearchCV(estimator=pipe, param_grid={'svm__C': C, 'svm__kernel': ['rbf', 'poly', 'linear'], 'svm__deg
                              scoring=['accuracy','f1','precision','recall','roc_auc'], cv=skf,
                              refit='roc_auc', verbose=1, return_train_score=True)
         svm_2feat.fit(X_train_2feat, Y_train.ravel())
         best_svm_2feat = svm_2feat.best_estimator_
         svm_2feat.best_params_
```

```
Fitting 5 folds for each of 24 candidates, totalling 120 fits
```

Out[44]:
```
{'svm__C': 0.5, 'svm__degree': 2, 'svm__kernel': 'rbf'}
```

In [45]:
```
print_stats(best_svm_2feat, x_test_2feat, y_test)
```

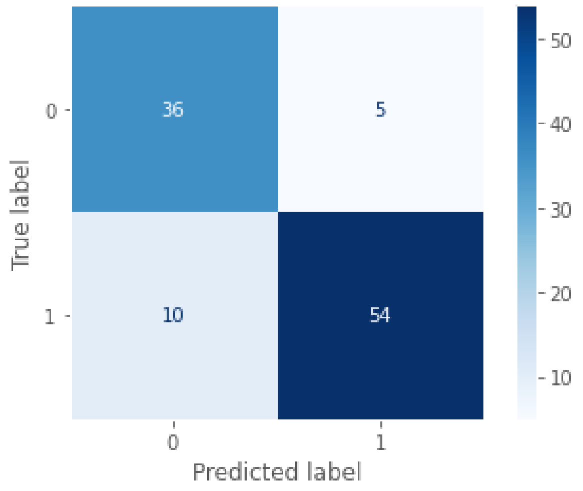
	AUROC	Accuracy	F1	Loss	Sensitiviy	Specificity
Performance statistics	0.866997	0.857143	0.878049	0.285714	0.84375	0.878049



Random Forest Classifier:

In [46]:
```
rfc_2feat = RandomForestClassifier(max_depth=10, random_state=0, criterion='gini')
rfc_2feat.fit(X_train_2feat, Y_train.ravel())
print_stats(rfc_2feat, x_test_2feat, y_test)
```

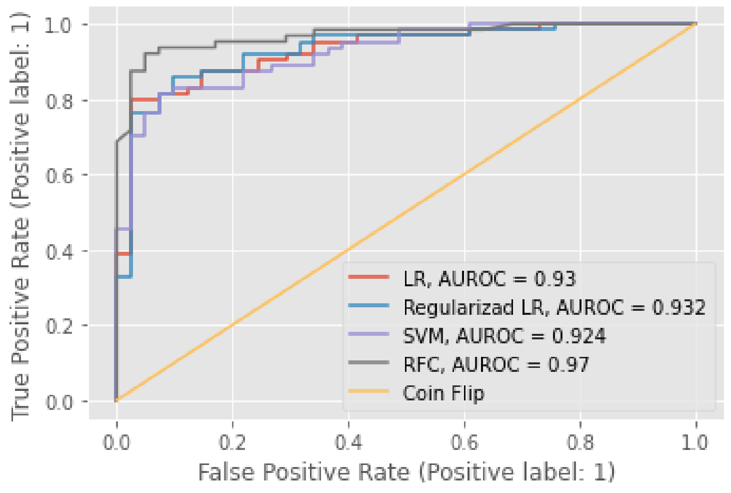
	AUROC	Accuracy	F1	Loss	Sensitiviy	Specificity
Performance statistics	0.893674	0.857143	0.878049	0.285714	0.84375	0.878049



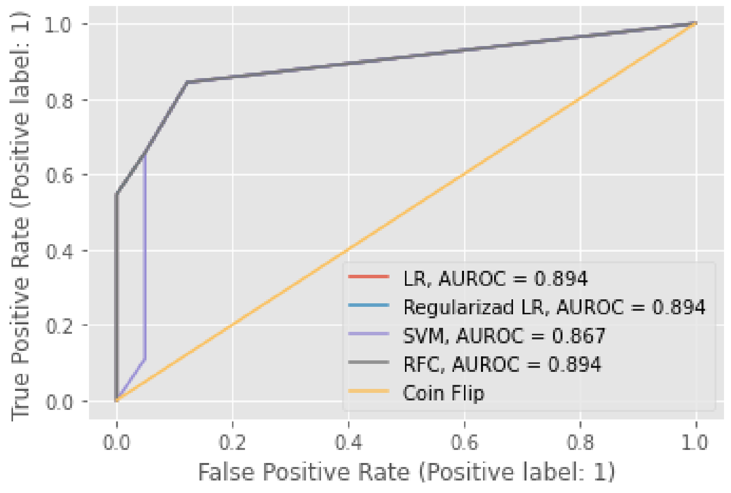
Now we compare the models based either on reduced-dimensionality data or on the two most indicative features.

In [47]:
```
print('\nROC of models using the reduced dimensionality data: \n')
classifiers = [lr_pca, lr_regul_pca, best_svm_pca, rfc_pca]
roc_score = []
plt.figure()
ax = plt.gca()
for clf in classifiers:
    plot_roc_curve(clf, x_test_pca, y_test, ax=ax)
    roc_score.append(np.round_(roc_auc_score(y_test, clf.predict_proba(x_test_pca)[:,1]), decimals=3))
ax.plot(np.linspace(0,1,x_test_pca.shape[0]),np.linspace(0,1,x_test_pca.shape[0]))
plt.legend(('LR, AUROC = '+str(roc_score[0]),'Regularizad LR, AUROC = '+str(roc_score[1]),
            'SVM, AUROC = '+str(roc_score[2]),'RFC, AUROC = '+str(roc_score[3]),'Coin Flip'))
plt.show()


print('\nROC of models using the most indicative two features: \n')
classifiers = [lr_2feat, lr_regul_2feat, best_svm_2feat, rfc_2feat]
roc_score = []
plt.figure()
ax = plt.gca()
for clf in classifiers:
    plot_roc_curve(clf, x_test_2feat, y_test, ax=ax)
    roc_score.append(np.round_(roc_auc_score(y_test, clf.predict_proba(x_test_2feat)[:,1]), decimals=3))
ax.plot(np.linspace(0,1,x_test_2feat.shape[0]),np.linspace(0,1,x_test_2feat.shape[0]))
plt.legend(('LR, AUROC = '+str(roc_score[0]),'Regularizad LR, AUROC = '+str(roc_score[1]),
            'SVM, AUROC = '+str(roc_score[2]),'RFC, AUROC = '+str(roc_score[3]),'Coin Flip'))
plt.show()
```

```
ROC of models using the reduced dimensionality data:
```



```
ROC of models using the most indicative two features:
```



In [48]:
```
print('\nPerformance statistics of models using the reduced dimensionality data: \n')

model_evaluation_metrics_pca = pd.DataFrame({'Linear': return_stats(lr_pca, x_test_pca, y_test),
                                         'Linear Regularized': return_stats(lr_regul_pca, x_test_pca, y_test),
                                         'Polynomial SVM': return_stats(best_svm_pca, x_test_pca, y_test),
                                         'Random Forest Classifier': return_stats(rfc_pca, x_test_pca, y_test)}).trans
display(model_evaluation_metrics_pca)

print('\nPerformance statistics of models using the most indicative two features: \n')

model_evaluation_metrics_2feat = pd.DataFrame({'Linear': return_stats(lr_2feat, x_test_2feat, y_test),
                                         'Linear Regularized': return_stats(lr_regul_2feat, x_test_2feat, y_test),
                                         'Polynomial SVM': return_stats(best_svm_2feat, x_test_2feat, y_test),
                                         'Random Forest Classifier': return_stats(rfc_2feat, x_test_2feat, y_test)}).
display(model_evaluation_metrics_2feat)
```

```
Performance statistics of models using the reduced dimensionality data:
```

	Loss	Sensitiviy	Specificity	Accuracy	F1	AUROC
Linear	0.369033	0.843750	0.853659	0.847619	0.870968	0.929878
Linear Regularized	0.329696	0.828125	0.902439	0.857143	0.876033	0.932165
Polynomial SVM	0.342857	0.890625	0.731707	0.828571	0.863636	0.924162
Random Forest Classifier	0.152381	0.906250	0.951220	0.923810	0.935484	0.969512

```
Performance statistics of models using the most indicative two features:
```

	Loss	Sensitiviy	Specificity	Accuracy	F1	AUROC
Linear	0.353078	0.84375	0.878049	0.857143	0.878049	0.893674
Linear Regularized	0.356137	0.84375	0.878049	0.857143	0.878049	0.893674
Polynomial SVM	0.285714	0.84375	0.878049	0.857143	0.878049	0.866997
Random Forest Classifier	0.285714	0.84375	0.878049	0.857143	0.878049	0.893674

From the ROC curves and the performance statistics above, we conclude that using PCA reduced dimensionality data leads to better results than selecting the two most indicative features. These results match our expectations as PCA generates new features based on combinations of all features, including the two most indicative features. The new features returned from PCA are optimal to the projection in terms of maximal variance.