HW #2 – Machine Learning in Healthcare 336546

Theory Questions

Question 1

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.

Answer 1

Model Accuracy is the ratio of number of correct predictions to the total number of input samples. It works well only if there are equal number of samples belonging to each class. For example, consider that there are 98% samples of class A and 2% samples of class B in our training set. Then our model can easily get 98% training accuracy by simply predicting every training sample belonging to class A. When the same model is tested on a test set with 60% samples of class A and 40% samples of class B, then the test accuracy would drop down to 60%. Classification Accuracy is great, but gives us the false sense of achieving high accuracy.

Model performance (F1 Score) is the Harmonic Mean between precision and recall. The F1 Score tries to find the balance between precision and recall. F1 Score tells how precise our classifier is (how many instances it classifies correctly), as well as how robust it is (it does not miss a significant number of instances). The range for F1 Score is [0, 1] and the greater the score, the better is the performance of our model. Therefor, model performance is more important to us.

For example T1D has an estimated ratio between 1 / 500-1000 in people under the age of 19. So if we will check a stupid classifier that classify everyone as negative to T1D on the general population we will get 99.8% accuracy... but there is value to such classifier.

Question 2

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.

Answer 2

Feature selection is the process of reducing the number of input variables when developing a predictive model. It is desirable to reduce the number of input variables to both reduce the computational cost of modeling and, in some cases, to improve the performance of the model. The central premise when using a feature selection technique is that the data contains some features that

are either redundant or irrelevant, and can thus be removed without incurring much loss of information. Redundant and irrelevant are two distinct notions, since one relevant feature may be redundant in the presence of another relevant feature with which it is strongly correlated.

On the one hand, a classifier trained using only 2 features has a greater chance to correctly identiffy T1D patients with a low chance to suffer from a heart attack. With only 2 features we significantly reduce the Degrees of freedom hence we are more robust and may even improve the False Positive probability. Yet, on the other, we might be missing the mutuale information and dependencies that other features may supply. Hence, the chances for us the miss-classify (False Positive) are increasing.

While a classifier that uses all of the features available may provide the most accurate classification, and preservs the mutual information and dependencies that features have on one another, it is not cost and time effective. Many features might be redundant or irrelevant for the classification task.

Question 3

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.

Answer 3

Logistic regression and support vector machines are supervised machine learning algorithms used to solve classification problems. Generally, it is usually advisable to first try to use logistic regression to see how the model does. If it fails, then we can try using linear and nonlinear SVM. Logistic regression and linear SVM have similar performance but depending on our features, one may be more efficient than the other.

Depending on the size of the training set (number of features and training examples) that we have, we can choose to use either logistic regression or SVM:

1. If the number of features is large while the number of training examples is small, we prefer to use logistic regression or linear SVM.
2. If the number of features is small while the number of training examples is intermediate, we can use nonlinear SVM.

Another aspect that we need to consider is the geometry of our date. If we have reason to believe theat the data won't be linearly separable (or need to be more robust to outliers) than we need to use nonlinear SVM. Otherwise, we need to try logistic regression first and see how the model perform. If logistic regression is not good enough, we can try linear SVM.

Question 4

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

Answer 4

Logistic regression (LR) focuses on maximizing the probability of the data. The farther the data lies from the separating hyperplane (on the correct side), the more accurate the LR is. Logistic regression assumes that the predictors aren't sufficient to determine the response variable, but determine a probability that is a logistic function of a linear combination of them. Logistic regression is great in a low number of dimensions and when the predictors don't suffice to give more than a probabilistic estimate of the response. Most important hyperparameters of Logistic regression are:

1. Solver - This parameter can take few values such as 'newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'. For small datasets, 'liblinear' is a good choice, whereas 'sag' and 'saga' are faster for large ones. For multiclass problems, only 'newton-cg', 'sag', 'saga' and 'lbfgs' handle multinomial loss
2. Penalty - This parameter can take values such as 'l1', 'l2', 'elasticnet', 'none'. Penalty is used to specify the norm used in the penalization. The 'newton-cg', 'sag' and 'lbfgs' solvers support only l2 penalties. 'elasticnet' is only supported by the 'saga' solver. If 'none' (not supported by the liblinear solver), no regularization is applied.
3. C - The C parameter controls the penality strength, which can also be effective. For small values of C, we increase the regularization strength which will create simple models which underfit the data. For big values of C, we low the power of regularization which implies the model and allow it to increase it's complexity, and therefore, overfit the data
4. class_weightdict or 'balanced' - If the classes are imbalance in the data then this parameter is used. The "balanced" mode uses the values of y to automatically adjust weights inversely proportional to class frequencies in the input data.
5. multi_class - This parameter takes values 'auto', 'ovr', 'multinomial'.

An SVM tries to find the separating hyperplane that maximizes the distance of the closest points to the margin (the support vectors). SVMs get's the best separating hyperplane, and they're efficient in high dimensional spaces. They're similar to regularization in terms of trying to find the lowest-normed vector that separates the data, but with a margin condition that favors choosing a good hyperplane. Additionally, SVMs only consider points near the margin (support vectors) while LR considers all the points in the data set. SVMs do better when there's a higher number of dimensions, and especially on problems where the predictors do certainly (or near-certainly) determine the responses. In order to improve the SVM model accuracy, there are several parameters need to be tuned. Three major parameters including:

1. Kernels: The main function of the kernel is to take low dimensional input space and transform it into a higher-dimensional space. It is mostly useful in non-linear separation problem.
2. C (Regularisation): C is the penalty parameter, which represents misclassification or error term. The misclassification or error term tells the SVM optimisation how much error is bearable. This is how you can control the trade-off between decision boundary and misclassification term. When C is high it will classify all the data points correctly, also there is a chance to overfit.
3. Gamma: It defines how far influences the calculation of plausible line of separation. When gamma is higher, nearby points will have high influence; low gamma means far away points also be considered to get the decision boundary.

Coding Assignment

Task 1: Load the data. Explain any preprocessing

The T1D database is an CSV file which holds the information of 565 patients. The file was sent by the nuse who collected the data and he/she informed us of missing values. Before we even start dealing with the data itself, we should apply the first and most important rule of data/signal processing: ALWAYS LOOK AND UNDERSTAND THE DATA FIRST! In order to do that, we will load the file into a variable called T1D_features and use descriptive statistics and visualization tools

In [2]:
```
import pandas as pd
import numpy as np
from pathlib import Path
import random
%load_ext autoreload

file = Path.cwd().joinpath('HW2_data.csv') # concatenates HW2_data.csv to the current fo
T1D_features = pd.read_csv(file)
random.seed(10)
T1D_features
```

Out[2]:

	Age	Gender	Increased Urination	Increased Thirst	Sudden Weight Loss	Weakness	Increased Hunger	Genital Thrush	Visual Blurring	Itching	Irrital
0	45	Male	No	No	No	Yes	No	No	No	Yes	
1	42	Male	No	No	No	No	No	No	No	No	
2	45	Male	Yes	Yes	No	Yes	No	Yes	No	No	
3	59	Female	No	No	No	No	No	No	No	No	
4	40	Female	Yes	Yes	Yes	Yes	No	No	Yes	Yes	
...	...	...	...	...	...	...	...	...	...	...	...
560	54	Male	Yes	Yes	Yes	Yes	No	NaN	Yes	Yes	
561	32	Male	No	No	No	No	No	NaN	No	No	
562	61	Female	Yes	No	No	No	Yes	No	No	No	
563	46	Male	No	No	No	Yes	No	No	No	Yes	
564	37	Male	No	No	No	No	No	No	No	No	

565 rows × 18 columns

As we can see, each column has it's unique values - either numbers or yes/no answers. Hence, we can easily check for any typing errors in our data. We will create a function for this purpose exactly: replce typing errors with NaN. The input of this function will be "T1D_features" (Dataframe) and the output will be a cleard Dataframe "c_t1d" (without type errors)

In [3]:
```
from Clean_Data import rm_typeo_and_miss

c_t1d = rm_typeo_and_miss(T1D_features)
c_t1d
```

```
D:\PycharmProjects\School\ML in Healthcare\HW 2\Clean_Data.py:33: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  c_t1d[column][i] = np.nan
```

Out[3]:

	Age	Gender	Increased Urination	Increased Thirst	Sudden Weight Loss	Weakness	Increased Hunger	Genital Thrush	Visual Blurring	Itching	Irrital
0	45.0	Male	No	No	No	Yes	No	No	No	Yes	
1	42.0	Male	No	No	No	No	No	No	No	No	
2	45.0	Male	Yes	Yes	No	Yes	No	Yes	No	No	
3	59.0	Female	No	No	No	No	No	No	No	No	
4	40.0	Female	Yes	Yes	Yes	Yes	No	No	Yes	Yes	
...	...	...	...	...	...	...	...	...	...	...	
560	54.0	Male	Yes	Yes	Yes	Yes	No	NaN	Yes	Yes	
561	32.0	Male	No	No	No	No	No	NaN	No	No	
562	61.0	Female	Yes	No	No	No	Yes	No	No	No	
563	46.0	Male	No	No	No	Yes	No	No	No	Yes	
564	37.0	Male	No	No	No	No	No	No	No	No	

565 rows × 18 columns

Now we can easily compare and make sure that our function works well by comparing the counts in each column:

In [4]:

```
print('NaN count for T1D_features dataframe')
print(T1D_features.isnull().sum())
```

```
NaN count for T1D_features dataframe
Age                    0
Gender                 0
Increased Urination    0
Increased Thirst      20
Sudden Weight Loss     9
Weakness               0
Increased Hunger      13
Genital Thrush        14
Visual Blurring        0
Itching               11
Irritability           0
Delayed Healing        0
Partial Paresis        0
Muscle Stiffness      15
Hair Loss              0
Obesity                0
Diagnosis              0
```

```
Family History          0
dtype: int64
```

In [5]:
```
print('NaN count for c_t1d dataframe')
print(c_t1d.isnull().sum())
```

```
NaN count for c_t1d dataframe
Age                     0
Gender                  0
Increased Urination     0
Increased Thirst       20
Sudden Weight Loss      9
Weakness                0
Increased Hunger       13
Genital Thrush         14
Visual Blurring         0
Itching                11
Irritability            0
Delayed Healing         0
Partial Paresis         0
Muscle Stiffness       15
Hair Loss               0
Obesity                 0
Diagnosis               0
Family History          0
dtype: int64
```

As we can see, there have been no type error in any of the features besides missing values. Now we can just as easily handle those missing values using random sampling of each series values. We will specify each replacing-values' probability, so that the random sampling will be derived from the same distribution as the original valid series data. We will use the function "nan2value_samp" to replaces each NaN with a sutable value for each feature using c_t1d as input. Afterwards, we will check again for any missing values:

In [6]:
```
from Clean_Data import nan2value_samp

c_samp = nan2value_samp(c_t1d)
print('NaN count for c_samp dataframe')
print(c_samp.isnull().sum())
```

```
NaN count for c_samp dataframe
Age                    0
Gender                 0
Increased Urination    0
Increased Thirst       0
Sudden Weight Loss     0
Weakness               0
Increased Hunger       0
Genital Thrush         0
Visual Blurring        0
Itching                0
Irritability           0
Delayed Healing        0
Partial Paresis        0
Muscle Stiffness       0
Hair Loss              0
Obesity                0
Diagnosis              0
Family History         0
dtype: int64
```

Task 2: Perform a test-train split of 20% test

In [7]:
```
from sklearn.model_selection import train_test_split

Diagnosis = c_samp[['Diagnosis']] # Our data has 2 classes: Negative and Positive
data = c_samp.drop(['Diagnosis'], axis=1)
X_train, X_test, y_train, y_test = train_test_split(data, np.ravel(Diagnosis), test_size
```

Task 3: Provide a detailed visualization and exploration of the data

An analysis to show that the distribution of the features is similar between test and train

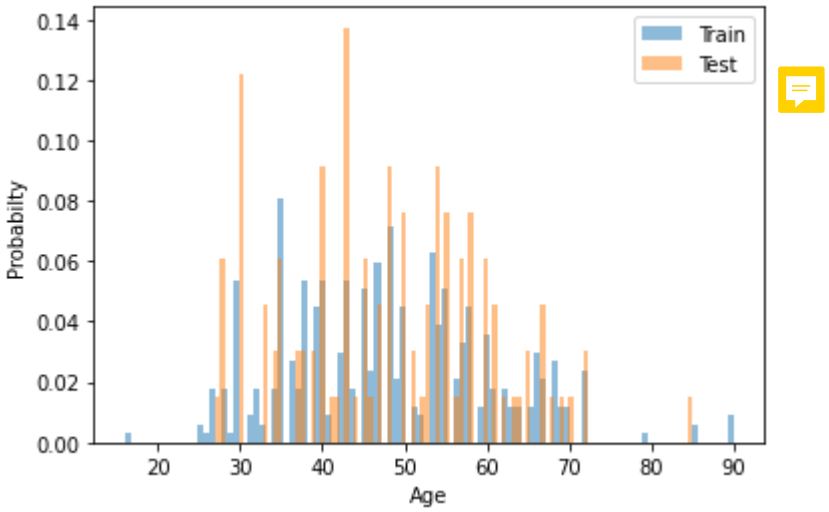
In [8]:
```
from Model_Prep import distribution_table

distribution_table(X_train, X_test, y_train, y_test)
```

Out[8]:

	Positive Feature	Train %	Test %	Delta %
0	Gender	64.0	65.0	1.0
1	Increased Urination	48.0	49.0	1.0
2	Increased Thirst	46.0	42.0	4.0
3	Sudden Weight Loss	40.0	46.0	6.0
4	Weakness	56.0	60.0	4.0
5	Increased Hunger	45.0	42.0	3.0
6	Genital Thrush	22.0	27.0	5.0
7	Visual Blurring	46.0	42.0	4.0
8	Itching	49.0	44.0	5.0
9	Irritability	23.0	26.0	3.0
10	Delayed Healing	47.0	43.0	4.0
11	Partial Paresis	43.0	42.0	1.0
12	Muscle Stiffness	37.0	39.0	2.0
13	Hair Loss	34.0	42.0	8.0
14	Obesity	17.0	14.0	3.0
15	Family History	50.0	52.0	2.0

In [16]:
```
import matplotlib.pyplot as plt
import seaborn as sns

bins = 100
feat = 'Age'
plt.hist(X_train[feat], bins, density=True, alpha=0.5, label='Train')
plt.hist(X_test[feat], bins, density=True, alpha=0.5, label='Test')
plt.xlabel(feat)
plt.ylabel('Probabilty')
```

```
plt.legend(loc='upper right')
plt.show()
```



Q1: What issues could an imbalance of features between train and test cause?

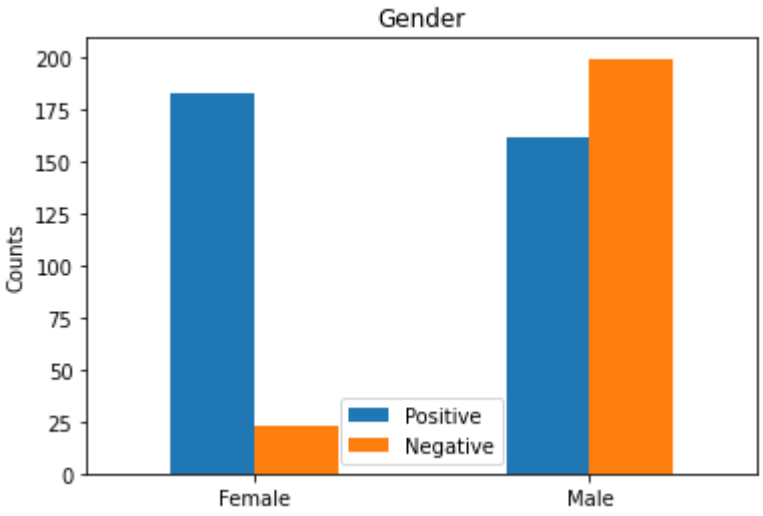
Q2: How could you solve the issue?

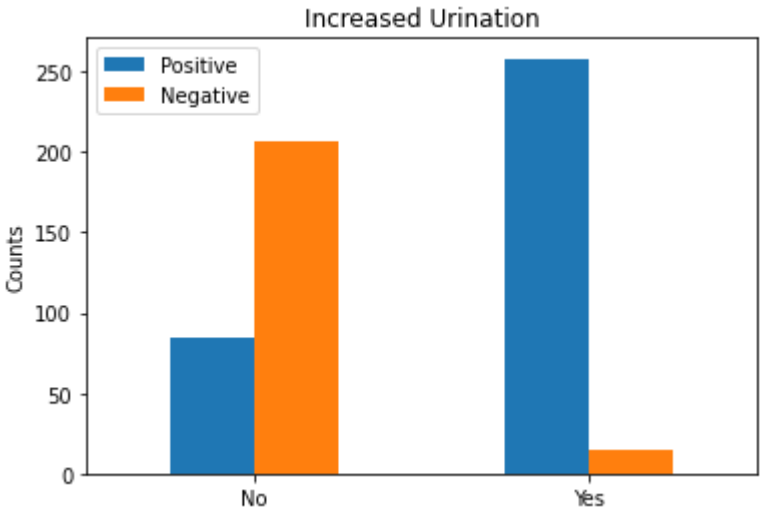
A1: An imbalance of features may cause an imbalanced weights and under-fit model. For example, an important feature with a rare apperence in the training set might be ignored or even be considerd as a feature with a strong correlation to another calss. And once we will evaluate the model, the accuracy will dramatically drop. The same phenomenon also happens when the imbalance is in favor of an insignificant feature.

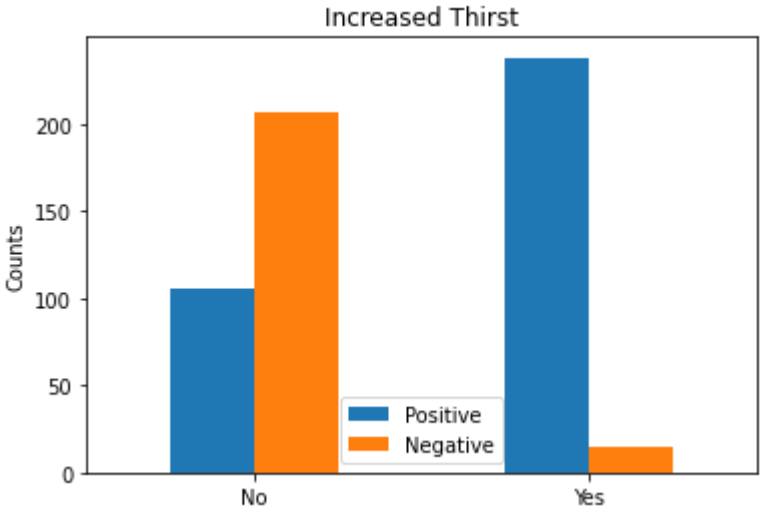
A2: Probably just by playing with ratio of train and test we might get a more suitable devision of the features. In the case of an imbalanced dataset, we should try re-sampling technique to get better results. Also, we can use a different metric for performance measurement such as F1 Score in case of imbalanced data set.

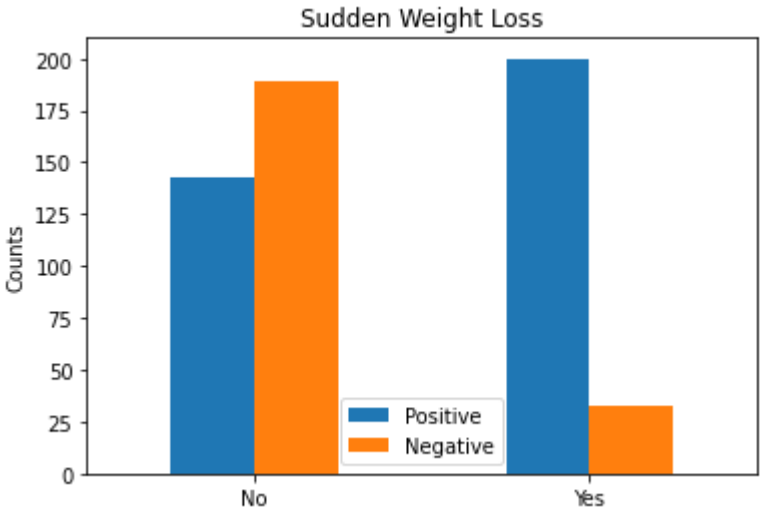
Plots to show the relationship between feature and label

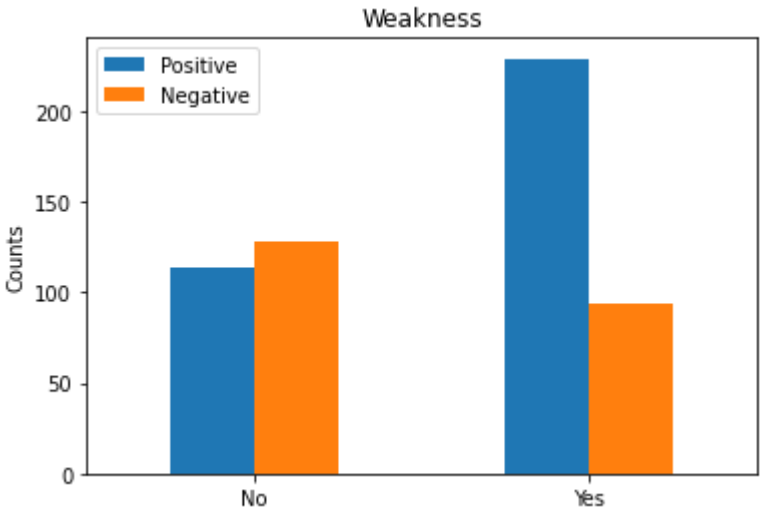
In [8]:
```
from Model_Prep import feature_label_plots

feature_label_plots(c_samp)
```

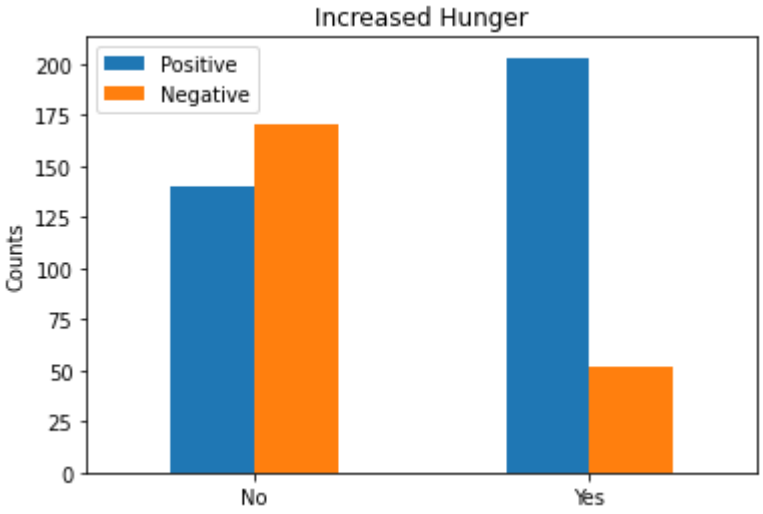

Gender
Counts
200
175
150
125
100
75
50
25
0
Positive
Negative
Female
Male

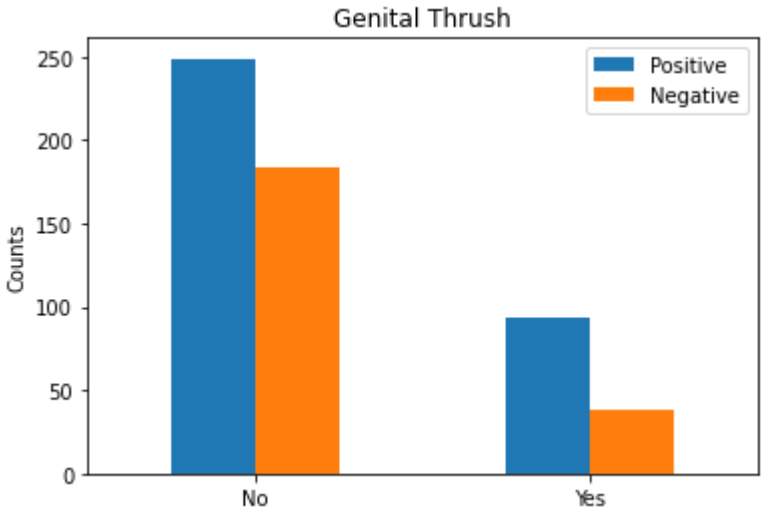

Increased Urination
Counts
250
200
150
100
50
0
Positive
Negative
No
Yes

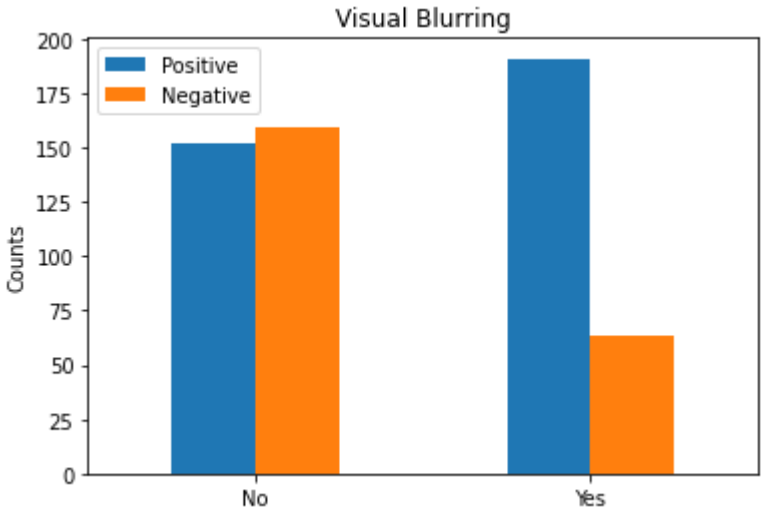

Increased Thirst
Counts
200
150
100
50
0
Positive
Negative
No
Yes

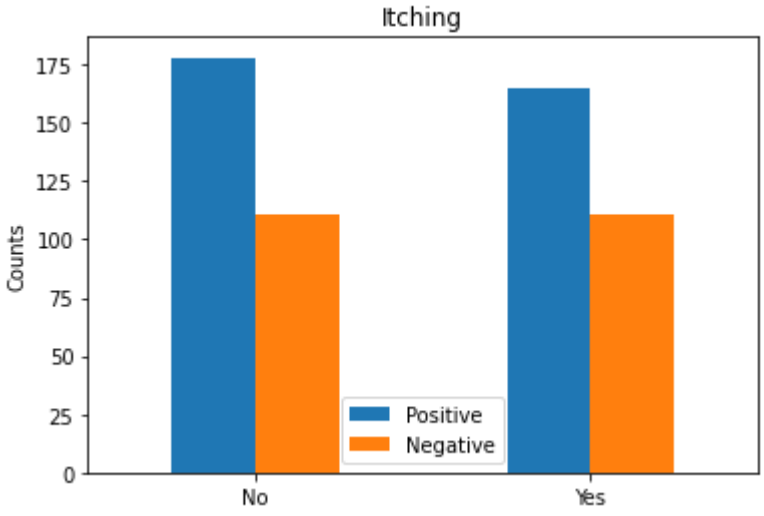

Sudden Weight Loss
Counts
200
175
150
125
100
75
50
25
0
Positive
Negative
No
Yes

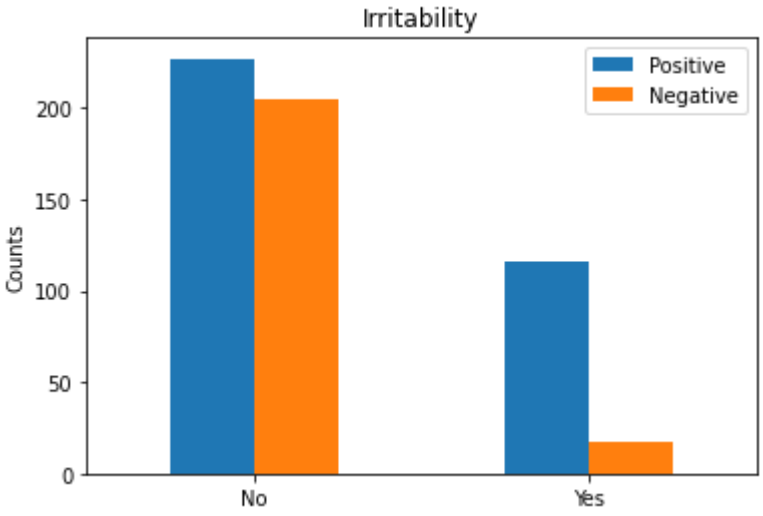

Weakness
Counts
Positive
Negative
200
150
100
50
0
No
Yes

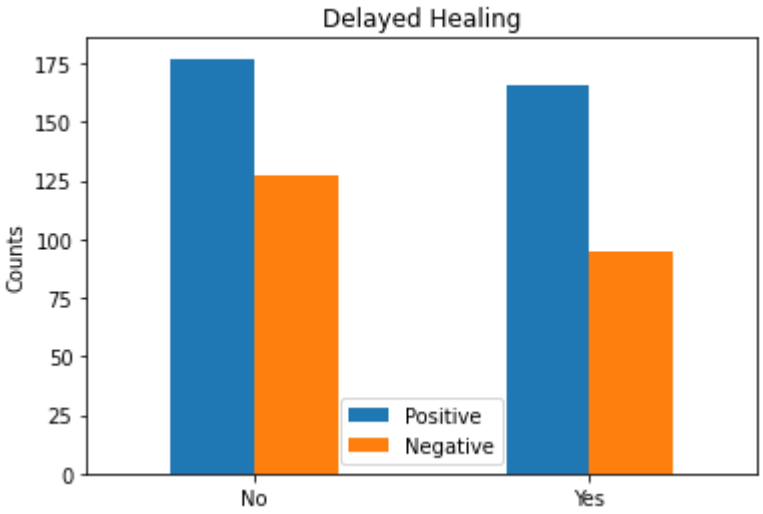

Increased Hunger
Counts
Positive
Negative
200
175
150
125
100
75
50
25
0
No
Yes

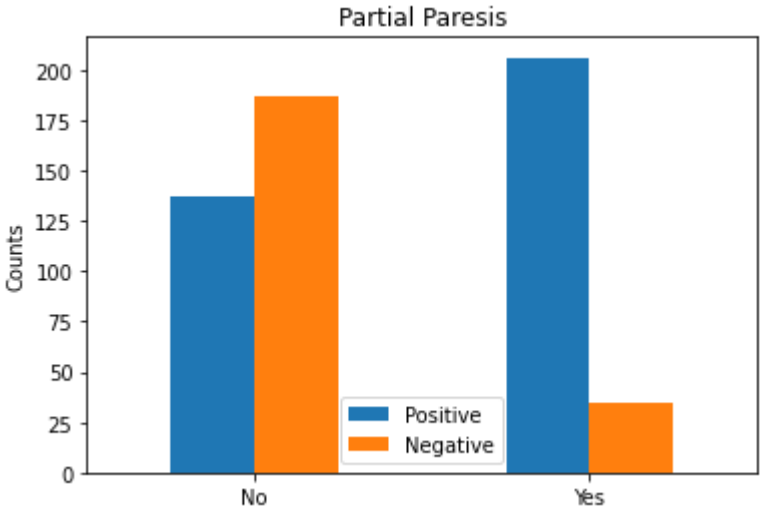

Genital Thrush
Positive
Negative
Counts
250
200
150
100
50
0
No
Yes

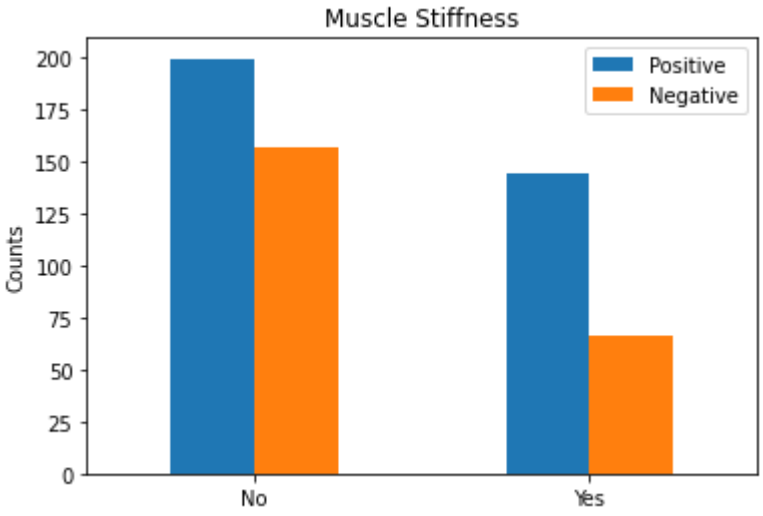

Visual Blurring
Positive
Negative
Counts
200
175
150
125
100
75
50
25
0
No
Yes

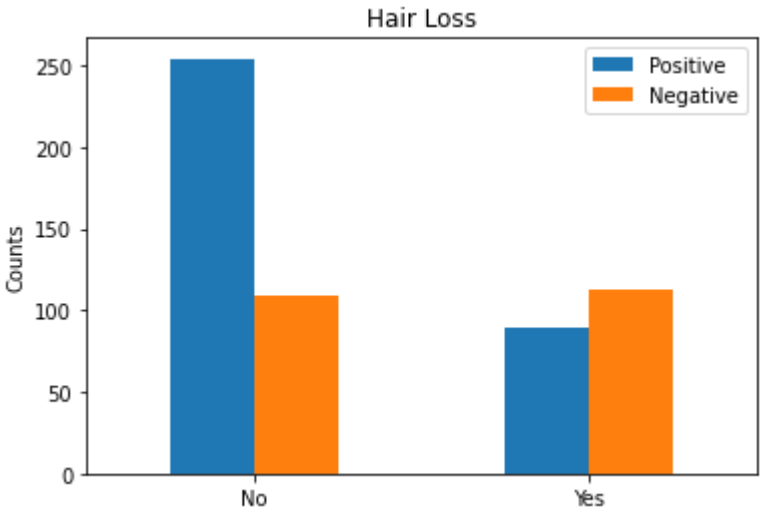

Itching
Positive
Negative
Counts
175
150
125
100
75
50
25
0
No
Yes

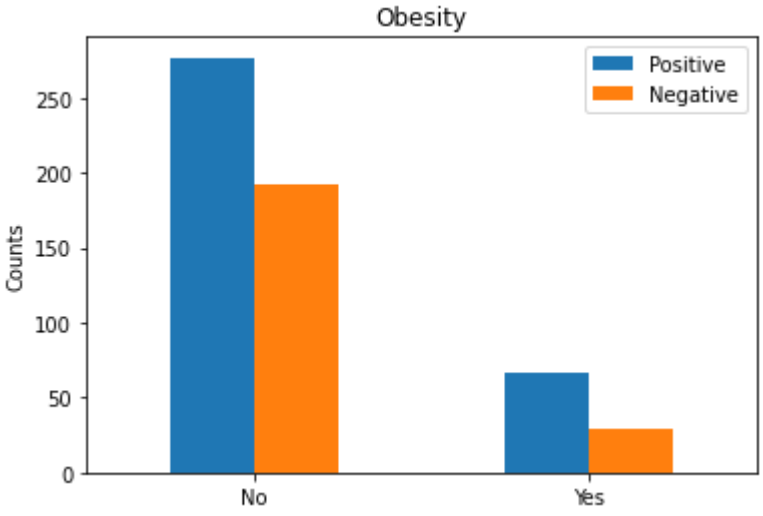

Irritability
Positive
Negative
Counts
200
150
100
50
0
No
Yes

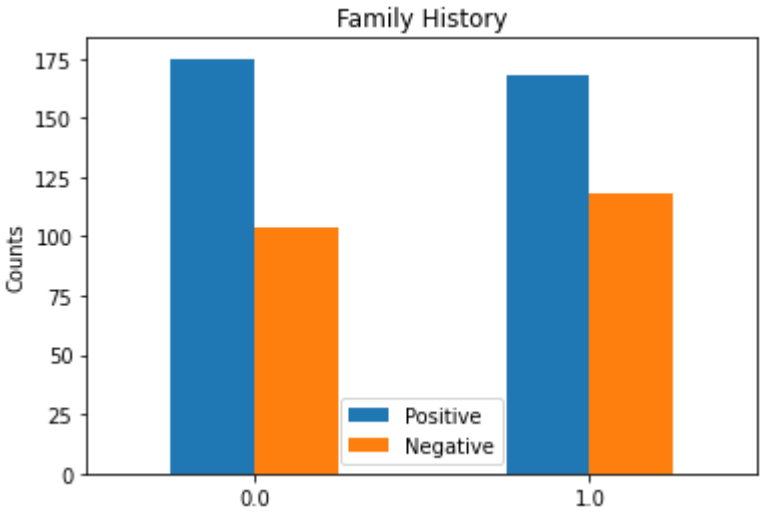

Delayed Healing
Counts
175
150
125
100
75
50
25
0
Positive
Negative
No
Yes


Partial Paresis
Counts
200
175
150
125
100
75
50
25
0
Positive
Negative
No
Yes


Muscle Stiffness
Positive
Negative
Counts
200
175
150
125
100
75
50
25
0
No
Yes


Hair Loss
Positive
Negative
Counts
250
200
150
100
50
0
No
Yes

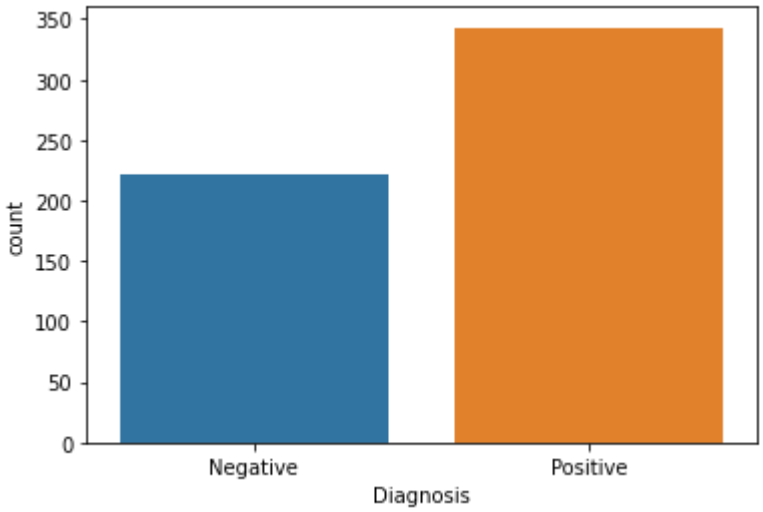

Obesity
Positive
Negative
Counts
250
200
150
100
50
0
No
Yes



Additional plots: Class Imbalance

In [9]:

```
import matplotlib.pyplot as plt
import seaborn as sns

g = sns.countplot(x = 'Diagnosis', data = Diagnosis)
g.set(xticklabels=['Negative', 'Positive'])
plt.show()
idx_1 = (Diagnosis == 'Positive').index[(Diagnosis == 'Positive')['Diagnosis'] == True].
idx_2 = (Diagnosis == 'Negative').index[(Diagnosis == 'Negative')['Diagnosis'] == True].

print("Negative samples account for " + str("{0:.2f}".format(100 * len(idx_2) / len(Diag
print("Positive samples account for " + str("{0:.2f}".format(100 * len(idx_1) / len(Diag
```



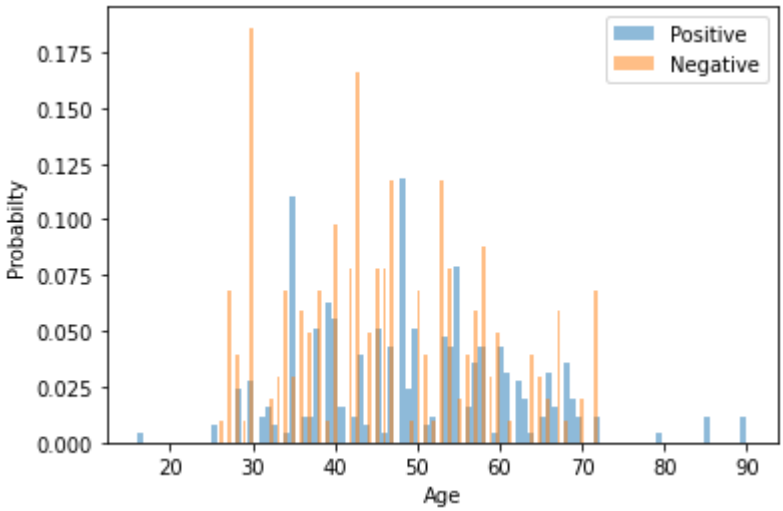

```
Negative samples account for 39.29% of the data.
Positive samples account for 60.71% of the data.
```

Additional plots: Features correlation with the labels and with one another

In [10]:

```
bins = 100
feat = 'Age'
plt.hist(data[feat].loc[idx_1], bins, density=True, alpha=0.5, label='Positive')
plt.hist(data[feat].loc[idx_2], bins, density=True, alpha=0.5, label='Negative')
plt.xlabel(feat)
plt.ylabel('Probabilty')
```

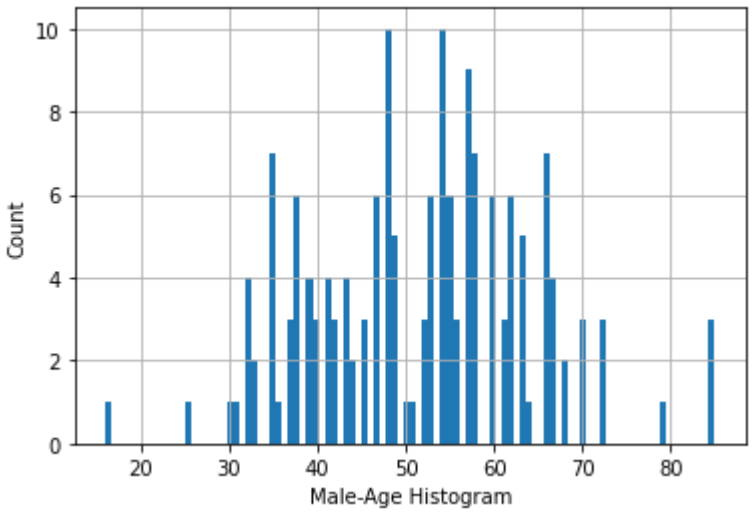
```
plt.legend(loc='upper right')
plt.show()
```

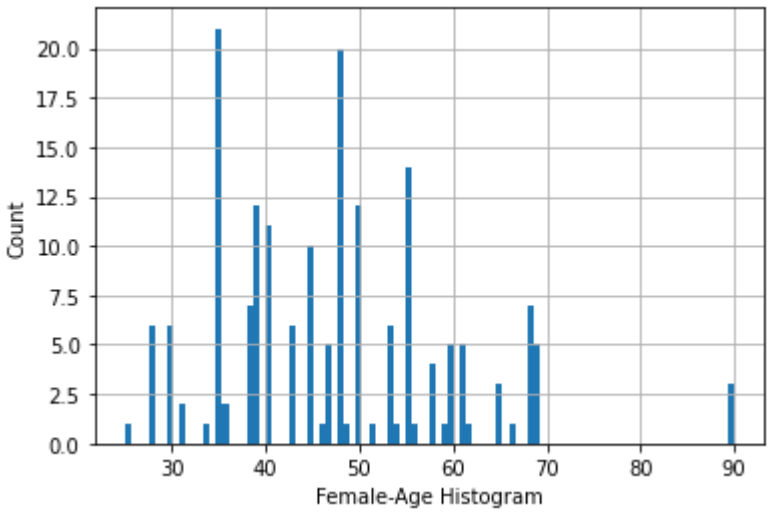


Additional plots: Male and female age histogram

In [11]:

```
Q_clean = c_samp['Age'].loc[(c_samp['Gender'] == 'Male') & (c_samp['Diagnosis'] == 'Posi
Q_clean.hist(bins=100)
plt.xlabel('Male-Age Histogram')
plt.ylabel('Count')
plt.show()

Q_clean = c_samp['Age'].loc[(c_samp['Gender'] == 'Female') & (c_samp['Diagnosis'] == 'Po
Q_clean.hist(bins=100)
plt.xlabel('Female-Age Histogram')
plt.ylabel('Count')
plt.show()
```





State any insights you have

Q1: Was there anything unexpected?

Q2: Are there any features that you feel will be particularly important to your model? Explain why.

A1: After looking at the distrubtion of P/N in a Gender dependent way, one can think that women has higher chance to get sick even due pervious studies determine that there is about 1.8:1 ratio of incidance in favor of men.

https://pubmed.ncbi.nlm.nih.gov/1541382/ A2: If we look closely on the bar plots from the previews task, we can easily target several features that have a strong correletion with the 'Positive' label: Gender (specifically female), Increased Urination and Thirst, Sudden Weight Loss and Hair Loss.

Task 4: Encode all your data as one hot vectors

In [13]:
```
from Model_Prep import encode_data

encoded_table, Diagnosis = encode_data(c_samp)
encoded_table
```

Out[13]:

	90.0	85.0	79.0	72.0	70.0	69.0	68.0	67.0	66.0	65.0	...	Genital Thrush	Visual Blurring	Itching	Irritability
0	0	0	0	0	0	0	0	0	0	0	...	0	0	1	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	1	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	1	1	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
560	0	0	0	0	0	0	0	0	0	0	...	1	1	1	1
561	0	0	0	0	0	0	0	0	0	0	...	1	0	0	1
562	0	0	0	0	0	0	0	0	0	0	...	0	0	0	1

	90.0	85.0	79.0	72.0	70.0	69.0	68.0	67.0	66.0	65.0	...	Genital Thrush	Visual Blurring	Itching	Irritability
563	0	0	0	0	0	0	0	0	0	0	...	0	0	1	0
564	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0

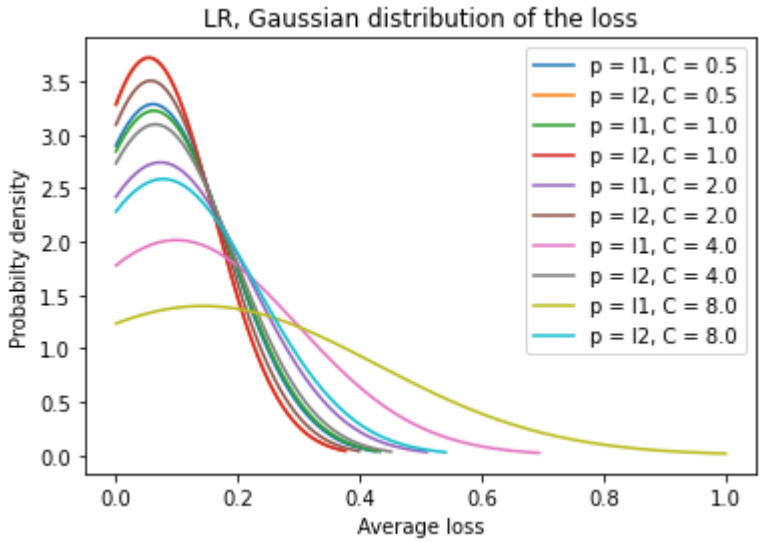
565 rows × 67 columns

Task 5: Choose, build and optimize Machine Learning Models

We will start with the most basic model - Linear Regression

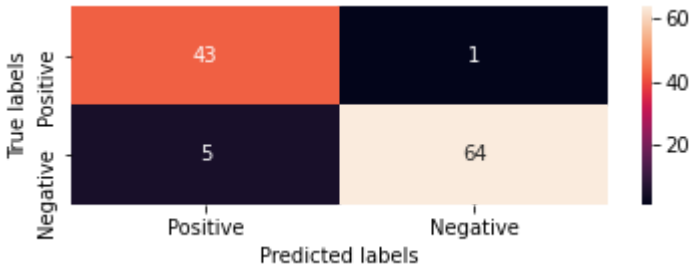
In [14]:
```
from sklearn import metrics
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from Model_Train import *


X_train, X_test, y_train, y_test = train_test_split(encoded_table, np.ravel(Diagnosis),
C = [0.5, 1.0, 2.0, 4.0, 8.0]
K = 5
val_dict = LR_cv_kfold(X_train, y_train, C=C, penalty=['l1', 'l2'], K=K)
```

In [15]:
```
import scipy.stats as stats
for d in val_dict:
    x = np.linspace(0, d['mu'] + 3 * d['sigma'], 1000)
    plt.plot(x,stats.norm.pdf(x, d['mu'], d['sigma']), label="p = " + d['penalty'] + ",
    plt.title('LR, Gaussian distribution of the loss')
    plt.xlabel('Average loss')
    plt.ylabel('Probabilty density')
plt.legend()
plt.show()
```



In [16]:
```
logreg = LogisticRegression(penalty='l1', C=8.0, solver='liblinear', max_iter=1000)
X_train, X_test, y_train, y_test = train_test_split(encoded_table, np.ravel(Diagnosis),
y_pred, w = pred_log(logreg, X_train, y_train, X_test)
```

```
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
ax1 = plt.subplot(211)
sns.heatmap(cnf_matrix, annot=True, xticklabels=['Positive','Negative'], yticklabels=['P
ax1.set(ylabel='True labels', xlabel='Predicted labels')
plt.show()
print("AUC score is: " + str("{0:.2f}".format(100 * metrics.roc_auc_score(y_test, y_pred
print("F1 score is: " + str("{0:.2f}".format(100 * metrics.f1_score(y_test, y_pred))) +
print("Loss is: " + str("{0:.2f}".format(metrics.log_loss(y_test, y_pred))))
print("Accuracy is: " + str("{0:.2f}".format(100 * metrics.accuracy_score(y_test, y_pred
```




```
AUC score is: 95.24%
F1 score is: 95.52%
Loss is: 1.83
Accuracy is: 94.69%
```

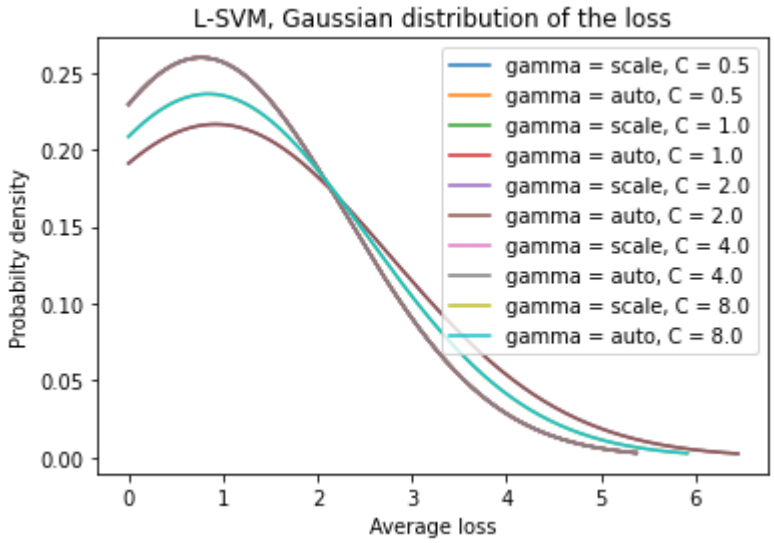
We will stay in the Linear domain, but with a different model - SVM

In [19]:

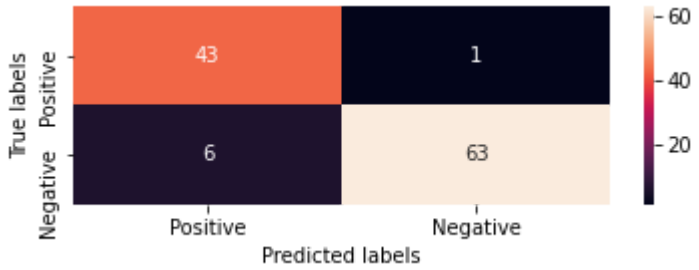
```
from sklearn.svm import SVC


X_train, X_test, y_train, y_test = train_test_split(encoded_table, np.ravel(Diagnosis),
C = [0.5, 1.0, 2.0, 4.0, 8.0]
gamma = ['scale', 'auto']
K = 5
val_dict = LSVM_cv_kfold(X_train, y_train, C=C, gamma=gamma, K=K)
```

In [20]:

```
import scipy.stats as stats
for d in val_dict:
    x = np.linspace(0, d['mu'] + 3 * d['sigma'], 1000)
    plt.plot(x,stats.norm.pdf(x, d['mu'], d['sigma']), label="gamma = " + d['gamma'] + "
    plt.title('L-SVM, Gaussian distribution of the loss')
    plt.xlabel('Average loss')
    plt.ylabel('Probabilty density')
plt.legend()
plt.show()
```



In [21]:
```
svclassifier = SVC(kernel='linear', gamma='auto', C=8.0)
svclassifier.fit(X_train, y_train)
y_pred = svclassifier.predict(X_test)

cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
ax1 = plt.subplot(211)
sns.heatmap(cnf_matrix, annot=True, xticklabels=['Positive','Negative'], yticklabels=['P
ax1.set(ylabel='True labels', xlabel='Predicted labels')
plt.show()
print("AUC score is: " + str("{0:.2f}".format(100 * metrics.roc_auc_score(y_test, y_pred
print("F1 score is: " + str("{0:.2f}".format(100 * metrics.f1_score(y_test, y_pred))) +
print("Loss is: " + str("{0:.2f}".format(metrics.log_loss(y_test, y_pred))))
print("Accuracy is: " + str("{0:.2f}".format(100 * metrics.accuracy_score(y_test, y_pred
```

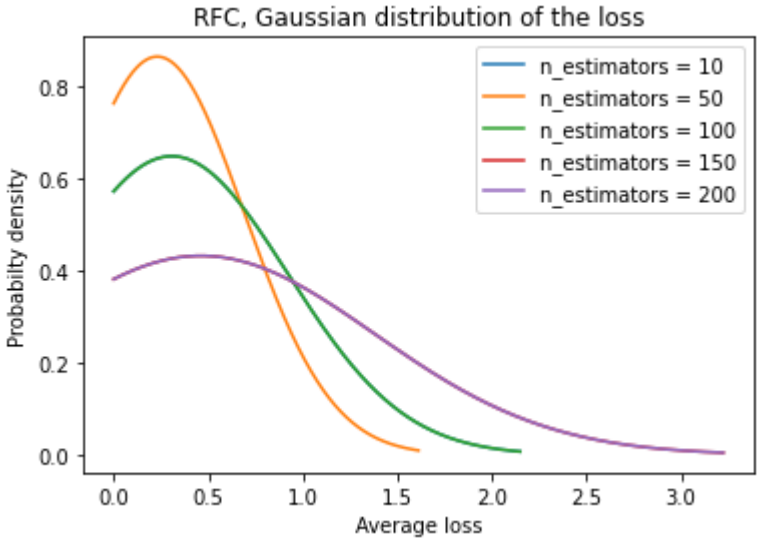



```
AUC score is: 94.52%
F1 score is: 94.74%
Loss is: 2.14
Accuracy is: 93.81%
```

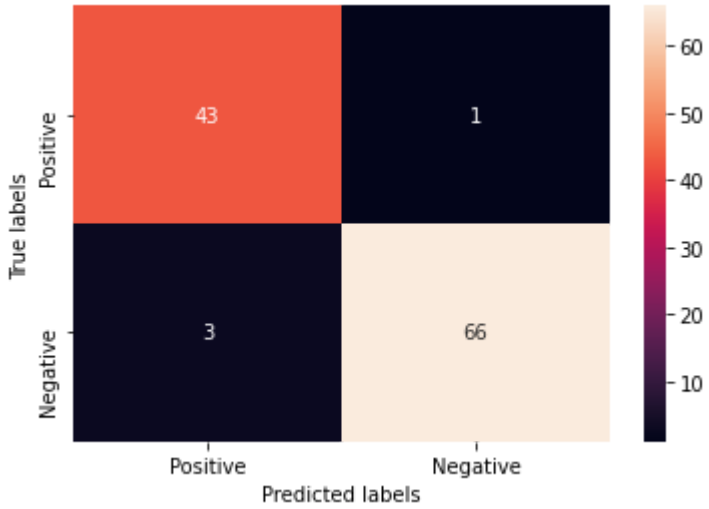
Now we can try non-linear model. We will start with the Random Forest Classifier

In [22]:
```
from sklearn.ensemble import RandomForestClassifier as rfc


X_train, X_test, y_train, y_test = train_test_split(encoded_table, np.ravel(Diagnosis),
n_estimators = [10, 50, 100, 150, 200]
clf = rfc(n_estimators=10)

K = 5
val_dict = RFC_cv_kfold(X_train, y_train, n_estimators=n_estimators, K=K)
```

In [23]:
```
import scipy.stats as stats
for d in val_dict:
    x = np.linspace(0, d['mu'] + 3 * d['sigma'], 1000)
    plt.plot(x,stats.norm.pdf(x, d['mu'], d['sigma']), label="n_estimators = " + str(d['
    plt.title('RFC, Gaussian distribution of the loss')
    plt.xlabel('Average loss')
    plt.ylabel('Probabilty density')
plt.legend()
plt.show()
```



In [24]:
```
from sklearn.ensemble import RandomForestClassifier as rfc

X_train, X_test, y_train, y_test = train_test_split(encoded_table, np.ravel(Diagnosis),
clf = rfc(n_estimators=100)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
ax = plt.subplot()
sns.heatmap(cnf_matrix, annot=True, xticklabels=['Positive', 'Negative'],
            yticklabels=['Positive', 'Negative'])
ax.set(ylabel='True labels', xlabel='Predicted labels')
plt.show()
print("AUC score is: " + str("{0:.2f}".format(100 * metrics.roc_auc_score(y_test, y_pred
print("F1 score is: " + str("{0:.2f}".format(100 * metrics.f1_score(y_test, y_pred))) +
print("Loss is: " + str("{0:.2f}".format(metrics.log_loss(y_test, y_pred))))
print("Accuracy is: " + str("{0:.2f}".format(100 * metrics.accuracy_score(y_test, y_pred
```

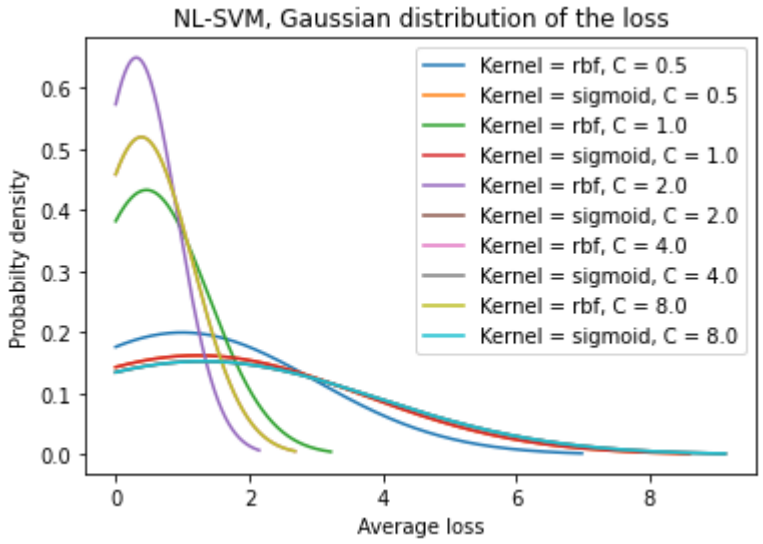



```
AUC score is: 96.69%
F1 score is: 97.06%
Loss is: 1.22
Accuracy is: 96.46%
```

We will try another non-linear model - SVM with a Gaussian and Sigmoid Kernels

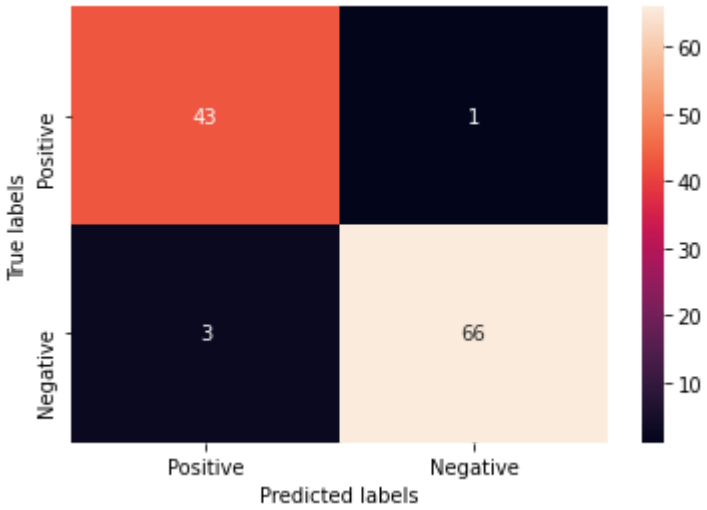
In [25]:
```
X_train, X_test, y_train, y_test = train_test_split(encoded_table, np.ravel(Diagnosis),
C = [0.5, 1.0, 2.0, 4.0, 8.0]
kernels = ['rbf', 'sigmoid']
K = 5
val_dict = NLSVM_cv_kfold(X_train, y_train, C=C, kernels=kernels, K=K)
```

In [26]:
```
import scipy.stats as stats
for d in val_dict:
    x = np.linspace(0, d['mu'] + 3 * d['sigma'], 1000)
    plt.plot(x,stats.norm.pdf(x, d['mu'], d['sigma']), label="Kernel = " + d['Kernel'] +
    plt.title('NL-SVM, Gaussian distribution of the loss')
    plt.xlabel('Average loss')
    plt.ylabel('Probabilty density')
plt.legend()
plt.show()
```



In [29]:

```
from sklearn.svm import SVC

svclassifier = SVC(kernel='rbf', C = 8.0)
X_train, X_test, y_train, y_test = train_test_split(encoded_table, np.ravel(Diagnosis),
svclassifier.fit(X_train, y_train)
y_pred = svclassifier.predict(X_test)
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
ax = plt.subplot()
sns.heatmap(cnf_matrix, annot=True, xticklabels=['Positive', 'Negative'],
            yticklabels=['Positive', 'Negative'])
ax.set(ylabel='True labels', xlabel='Predicted labels')
plt.show()
print("AUC score is: " + str("{0:.2f}".format(100 * metrics.roc_auc_score(y_test, y_pred
print("F1 score is: " + str("{0:.2f}".format(100 * metrics.f1_score(y_test, y_pred))) +
print("Loss is: " + str("{0:.2f}".format(metrics.log_loss(y_test, y_pred))))
print("Accuracy is: " + str("{0:.2f}".format(100 * metrics.accuracy_score(y_test, y_pred
```




```
AUC score is: 96.69%
F1 score is: 97.06%
Loss is: 1.22
Accuracy is: 96.46%
```

As we can see, non-linear models has the best results on our data. Although the linear models present lower performences on our data, the model evaluation metrices are not as inferior as we would expect.

Task 6: Feature Selection

In [33]:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import ExtraTreesClassifier


# Build a forest and compute the impurity-based feature importances
forest = ExtraTreesClassifier(n_estimators=250, random_state=0)

forest.fit(X_train, y_train)
importances = forest.feature_importances_
std = np.std([tree.feature_importances_ for tree in forest.estimators_],
             axis=0)
indices = np.argsort(importances)[::-1]

# Print the feature ranking
```

```
print("Feature ranking:")

for f in range(X_train.shape[1]):
    print("%d. %s (%f)" % (f + 1, X_train.columns[indices[f]], importances[indices[f]]))
```

```
Feature ranking:
1. Increased Urination (0.207147)
2. Increased Thirst (0.126840)
3. Gender (0.089465)
4. Partial Paresis (0.063002)
5. Sudden Weight Loss (0.046926)
6. Increased Hunger (0.040165)
7. Hair Loss (0.036708)
8. Irritability (0.034737)
9. Visual Blurring (0.033866)
10. Delayed Healing (0.030979)
11. Itching (0.030165)
12. Genital Thrush (0.027867)
13. Weakness (0.023379)
14. Muscle Stiffness (0.021144)
15. Obesity (0.017861)
16. 35.0 (0.011797)
17. 47.0 (0.008956)
18. 58.0 (0.008292)
19. Family History (0.007825)
20. 44.0 (0.006309)
21. 40.0 (0.006262)
22. 53.0 (0.006106)
23. 72.0 (0.006055)
24. 33.0 (0.005953)
25. 37.0 (0.005731)
26. 45.0 (0.005321)
27. 43.0 (0.005304)
28. 56.0 (0.004885)
29. 32.0 (0.004775)
30. 48.0 (0.004740)
31. 67.0 (0.004376)
32. 49.0 (0.004212)
33. 65.0 (0.003761)
34. 59.0 (0.003718)
35. 42.0 (0.003685)
36. 38.0 (0.003411)
37. 64.0 (0.003128)
38. 30.0 (0.003048)
39. 36.0 (0.002856)
40. 54.0 (0.002725)
41. 57.0 (0.002707)
42. 46.0 (0.002428)
43. 90.0 (0.002368)
44. 28.0 (0.002363)
45. 60.0 (0.002318)
46. 66.0 (0.002314)
47. 50.0 (0.002258)
48. 55.0 (0.002053)
49. 34.0 (0.001975)
50. 25.0 (0.001699)
51. 41.0 (0.001696)
52. 70.0 (0.001523)
53. 61.0 (0.001407)
54. 51.0 (0.001392)
55. 39.0 (0.001328)
56. 63.0 (0.001310)
57. 62.0 (0.001309)
58. 69.0 (0.001091)
```

```
59. 27.0 (0.000701)
60. 16.0 (0.000601)
61. 68.0 (0.000593)
62. 31.0 (0.000404)
63. 52.0 (0.000305)
64. 79.0 (0.000285)
65. 29.0 (0.000050)
66. 85.0 (0.000032)
67. 26.0 (0.000008)
```

Based on the random forest results, the 2 most important features are:

1. Increased Urination (0.214588)
2. Increased Thirst (0.125018)

These results match up exactly with the feature exploration we did. more than that - the results are matched up with the known clinical sympthoms of T1D

Task 7: Data Separability Visualization

Perform dimensionality reduction on the dataset so that you can plot your data in a 2d plot (show samples with positive and negative labels in different colors).

We will perform dimensionality reduction using the PCA Algorithm

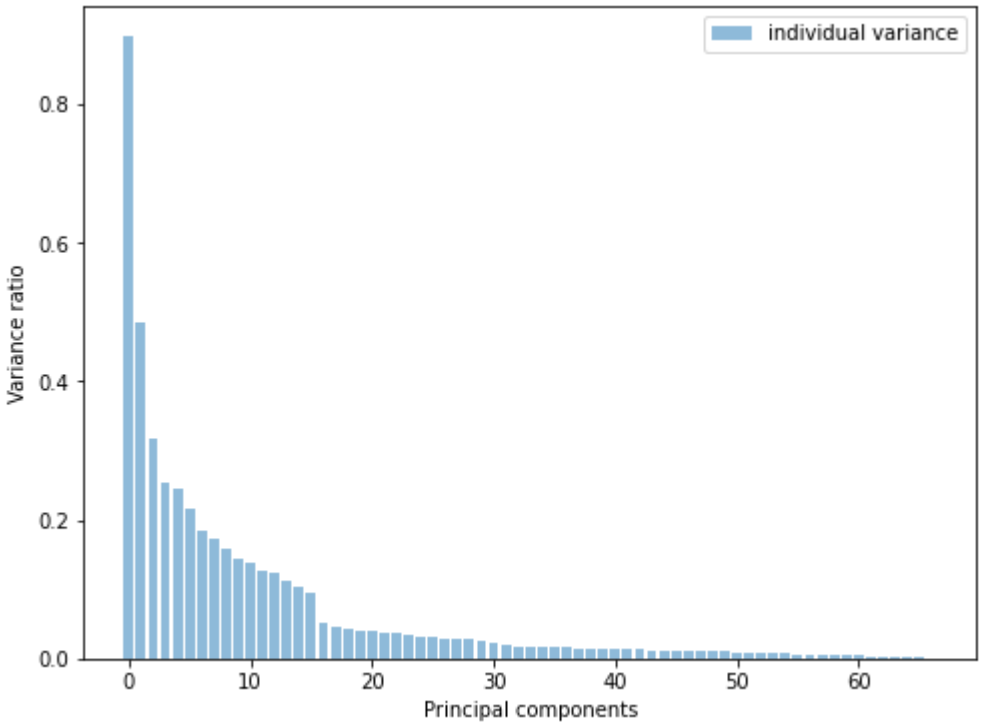
In [101…

```
# Machine learning systems work with integers, we need to encode these
# string characters into ints

encoded_table = c_samp.copy()
encoded_table = encoded_table.replace('Yes', 1)
encoded_table = encoded_table.replace('No', 0)
encoded_table = encoded_table.replace('Male', 1)
encoded_table = encoded_table.replace('Female', 0)
encoded_table = encoded_table.replace('Positive', 1)
encoded_table = encoded_table.replace('Negative', 0)

unique_ages = np.unique(encoded_table['Age'])
for elem in unique_ages:
    encoded_table.insert(0, str(elem), 0)
for idx, age in enumerate(encoded_table['Age']):
    encoded_table[str(age)][idx] = 1
encoded_table = encoded_table.drop('Age', axis=1)

Diagnosis = encoded_table['Diagnosis'].copy()
encoded_table = encoded_table.drop('Diagnosis', axis=1)
encoded_table = encoded_table.astype(int)
```

In [102…

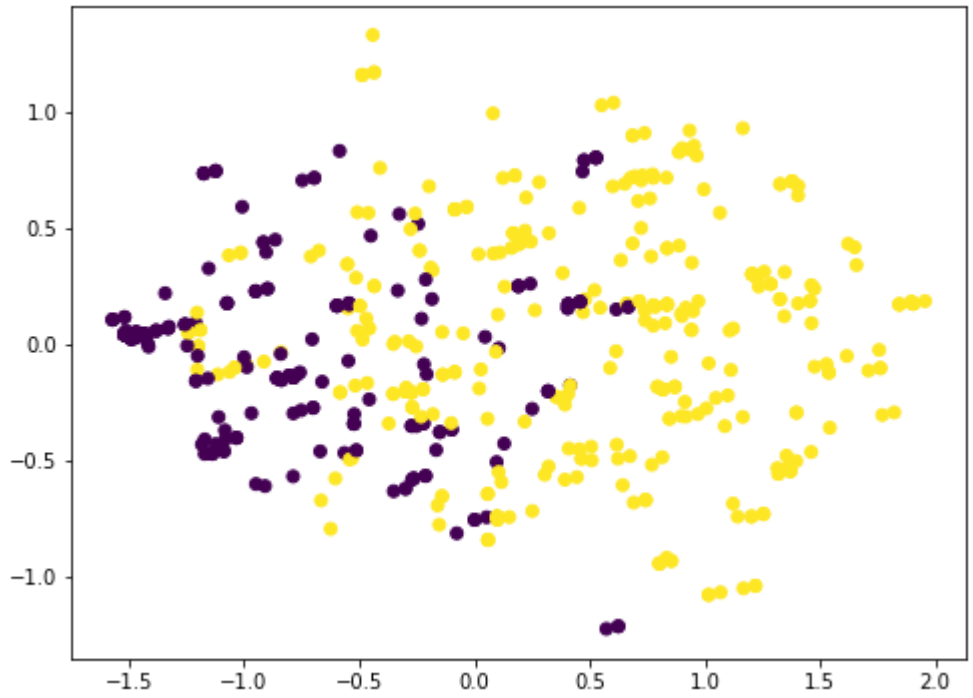
```
from sklearn.decomposition import PCA

pca = PCA()
pca.fit_transform(encoded_table)
pca_variance = pca.explained_variance_

plt.figure(figsize=(8, 6))
plt.bar(range(len(encoded_table.columns)), pca_variance, alpha=0.5, align='center', labe
plt.legend()
```

```
plt.ylabel('Variance ratio')
plt.xlabel('Principal components')
plt.show()
```

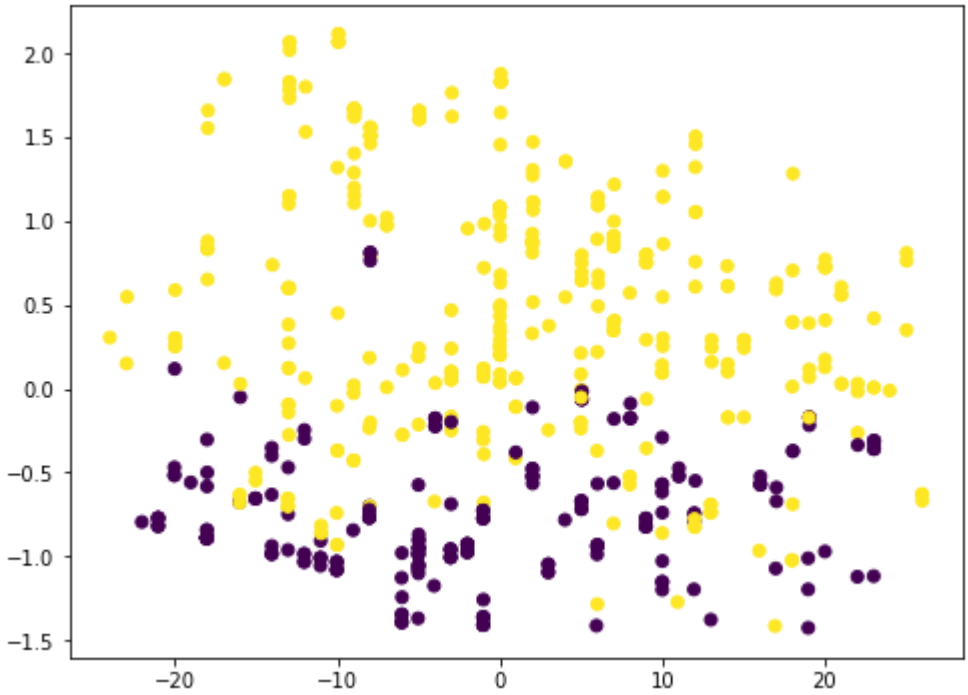


In [121…

```
pca2 = PCA(n_components=len(encoded_table.columns))
pca2.fit(encoded_table)
x_3d = pca2.transform(encoded_table)

plt.figure(figsize=(8,6))
plt.scatter(x_3d[:,0], x_3d[:,5], c=Diagnosis)
plt.show()
```



In [124…
```
pca3 = PCA(n_components=2) # convert the features into the 2 top features
pca3.fit(X_features)
x_3d = pca3.transform(X_features)

plt.figure(figsize=(8,6))
plt.scatter(x_3d[:,0], x_3d[:,1], c=Diagnosis)
plt.show()
```



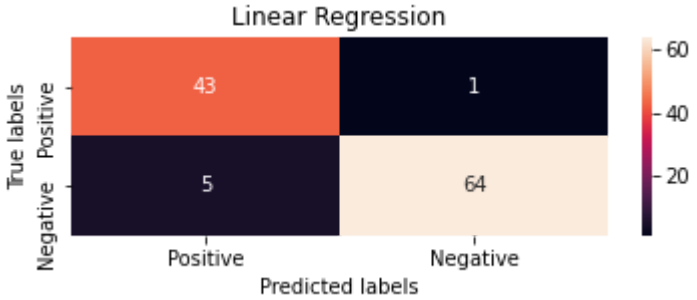
Perform dimensionality reduction on the dataset so that you can plot your data in a 2d plot (show samples with positive and negative labels in different colors).

As we can see, even when reduced to just two features, the data is sill not separable

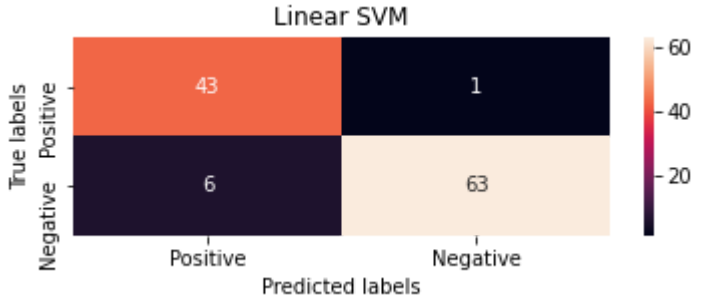
Train the same models above on the dimensionality-reduced training set

In [132…
```
pca2 = PCA(n_components=len(encoded_table.columns))
pca2.fit(encoded_table)
x_3d = pca2.transform(encoded_table)
X_train, X_test, y_train, y_test = train_test_split(x_3d, np.ravel(Diagnosis), test_size


## Linear Regression
logreg = LogisticRegression(penalty='l1', C=8.0, solver='liblinear', max_iter=1000)
y_pred, w = pred_log(logreg, X_train, y_train, X_test)
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
ax1 = plt.subplot(211)
sns.heatmap(cnf_matrix, annot=True, xticklabels=['Positive','Negative'], yticklabels=['P
ax1.set(ylabel='True labels', xlabel='Predicted labels', title='Linear Regression')
plt.show()
print("AUC score is: " + str("{0:.2f}".format(100 * metrics.roc_auc_score(y_test, y_pred
print("F1 score is: " + str("{0:.2f}".format(100 * metrics.f1_score(y_test, y_pred))) +
print("Loss is: " + str("{0:.2f}".format(metrics.log_loss(y_test, y_pred))))
print("Accuracy is: " + str("{0:.2f}".format(100 * metrics.accuracy_score(y_test, y_pred

## Linear SVM
svclassifier = SVC(kernel='linear', gamma='auto', C=8.0)
svclassifier.fit(X_train, y_train)
```

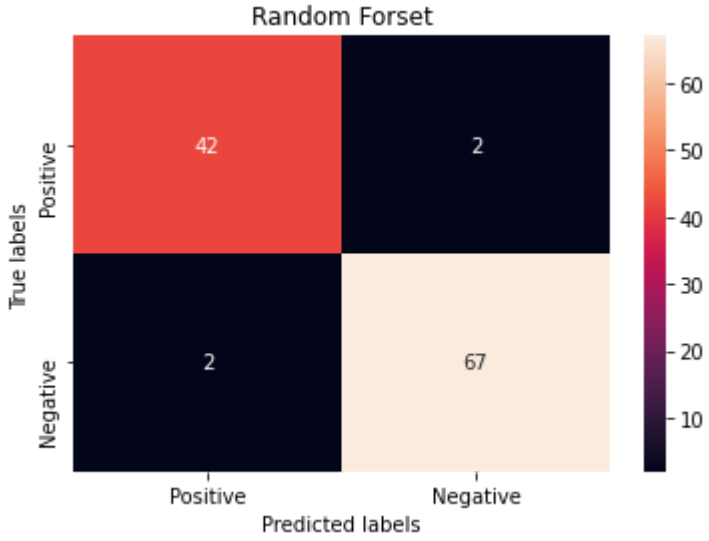
```
y_pred = svclassifier.predict(X_test)
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
ax1 = plt.subplot(211)
sns.heatmap(cnf_matrix, annot=True, xticklabels=['Positive','Negative'], yticklabels=['P
ax1.set(ylabel='True labels', xlabel='Predicted labels', title='Linear SVM')
plt.show()
print("AUC score is: " + str("{0:.2f}".format(100 * metrics.roc_auc_score(y_test, y_pred
print("F1 score is: " + str("{0:.2f}".format(100 * metrics.f1_score(y_test, y_pred))) +
print("Loss is: " + str("{0:.2f}".format(metrics.log_loss(y_test, y_pred))))
print("Accuracy is: " + str("{0:.2f}".format(100 * metrics.accuracy_score(y_test, y_pred

## Random Forest
clf = rfc(n_estimators=100)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
ax = plt.subplot()
sns.heatmap(cnf_matrix, annot=True, xticklabels=['Positive', 'Negative'],
            yticklabels=['Positive', 'Negative'])
ax.set(ylabel='True labels', xlabel='Predicted labels', title='Random Forset')
plt.show()
print("AUC score is: " + str("{0:.2f}".format(100 * metrics.roc_auc_score(y_test, y_pred
print("F1 score is: " + str("{0:.2f}".format(100 * metrics.f1_score(y_test, y_pred))) +
print("Loss is: " + str("{0:.2f}".format(metrics.log_loss(y_test, y_pred))))
print("Accuracy is: " + str("{0:.2f}".format(100 * metrics.accuracy_score(y_test, y_pred

## Non-Linear SVM
svclassifier = SVC(kernel='rbf', C = 8.0)
svclassifier.fit(X_train, y_train)
y_pred = svclassifier.predict(X_test)
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
ax = plt.subplot()
sns.heatmap(cnf_matrix, annot=True, xticklabels=['Positive', 'Negative'],
            yticklabels=['Positive', 'Negative'])
ax.set(ylabel='True labels', xlabel='Predicted labels', title='Non-Linear SVM')
plt.show()
print("AUC score is: " + str("{0:.2f}".format(100 * metrics.roc_auc_score(y_test, y_pred
print("F1 score is: " + str("{0:.2f}".format(100 * metrics.f1_score(y_test, y_pred))) +
print("Loss is: " + str("{0:.2f}".format(metrics.log_loss(y_test, y_pred))))
print("Accuracy is: " + str("{0:.2f}".format(100 * metrics.accuracy_score(y_test, y_pred
```



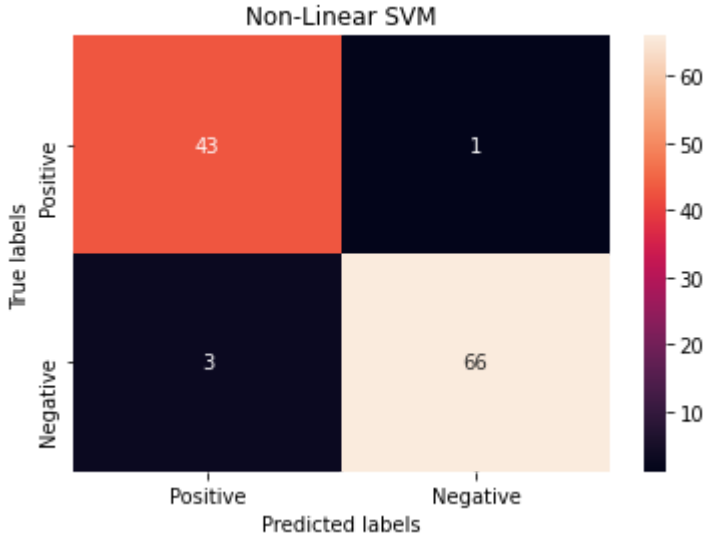

```
AUC score is: 95.24%
F1 score is: 95.52%
Loss is: 1.83
Accuracy is: 94.69%
```




```
AUC score is: 94.52%
F1 score is: 94.74%
Loss is: 2.14
Accuracy is: 93.81%
```




```
AUC score is: 96.28%
F1 score is: 97.10%
Loss is: 1.22
Accuracy is: 96.46%
```




```
AUC score is: 96.69%
F1 score is: 97.06%
Loss is: 1.22
Accuracy is: 96.46%
```

Train the same models on the best two features from section 6

Based on the random forest results, the 2 most important features are:

1. Increased Urination (0.214588)
2. Increased Thirst (0.125018)

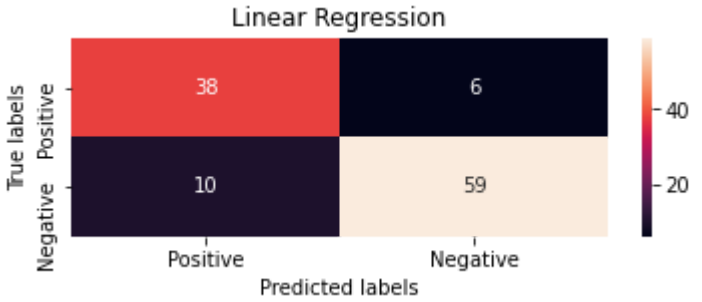
In [141…

```
encoded_table = encoded_table[['Increased Urination', 'Increased Thirst']]
X_train, X_test, y_train, y_test = train_test_split(encoded_table, np.ravel(Diagnosis),
```

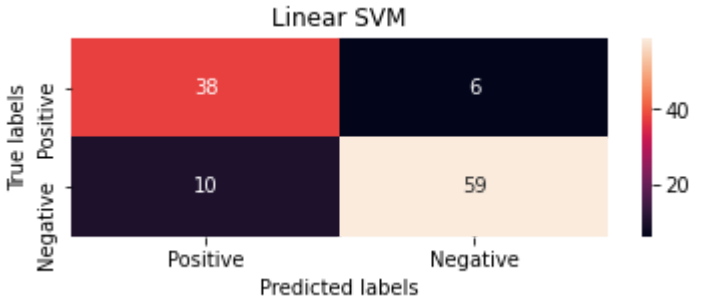
In [142…

```
## Linear Regression
logreg = LogisticRegression(penalty='l1', C=8.0, solver='liblinear', max_iter=1000)
y_pred, w = pred_log(logreg, X_train, y_train, X_test)
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
ax1 = plt.subplot(211)
sns.heatmap(cnf_matrix, annot=True, xticklabels=['Positive','Negative'], yticklabels=['P
ax1.set(ylabel='True labels', xlabel='Predicted labels', title='Linear Regression')
plt.show()
print("AUC score is: " + str("{0:.2f}".format(100 * metrics.roc_auc_score(y_test, y_pred
print("F1 score is: " + str("{0:.2f}".format(100 * metrics.f1_score(y_test, y_pred))) +
print("Loss is: " + str("{0:.2f}".format(metrics.log_loss(y_test, y_pred))))
print("Accuracy is: " + str("{0:.2f}".format(100 * metrics.accuracy_score(y_test, y_pred

## Linear SVM
svclassifier = SVC(kernel='linear', gamma='auto', C=8.0)
svclassifier.fit(X_train, y_train)
y_pred = svclassifier.predict(X_test)
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
ax1 = plt.subplot(211)
sns.heatmap(cnf_matrix, annot=True, xticklabels=['Positive','Negative'], yticklabels=['P
ax1.set(ylabel='True labels', xlabel='Predicted labels', title='Linear SVM')
plt.show()
print("AUC score is: " + str("{0:.2f}".format(100 * metrics.roc_auc_score(y_test, y_pred
print("F1 score is: " + str("{0:.2f}".format(100 * metrics.f1_score(y_test, y_pred))) +
print("Loss is: " + str("{0:.2f}".format(metrics.log_loss(y_test, y_pred))))
print("Accuracy is: " + str("{0:.2f}".format(100 * metrics.accuracy_score(y_test, y_pred

## Random Forest
clf = rfc(n_estimators=100)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
ax = plt.subplot()
sns.heatmap(cnf_matrix, annot=True, xticklabels=['Positive', 'Negative'],
            yticklabels=['Positive', 'Negative'])
ax.set(ylabel='True labels', xlabel='Predicted labels', title='Random Forset')
plt.show()
print("AUC score is: " + str("{0:.2f}".format(100 * metrics.roc_auc_score(y_test, y_pred
print("F1 score is: " + str("{0:.2f}".format(100 * metrics.f1_score(y_test, y_pred))) +
print("Loss is: " + str("{0:.2f}".format(metrics.log_loss(y_test, y_pred))))
print("Accuracy is: " + str("{0:.2f}".format(100 * metrics.accuracy_score(y_test, y_pred

## Non-Linear SVM
svclassifier = SVC(kernel='rbf', C = 8.0)
svclassifier.fit(X_train, y_train)
y_pred = svclassifier.predict(X_test)
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
ax = plt.subplot()
sns.heatmap(cnf_matrix, annot=True, xticklabels=['Positive', 'Negative'],
            yticklabels=['Positive', 'Negative'])
ax.set(ylabel='True labels', xlabel='Predicted labels', title='Non-Linear SVM')
plt.show()
```

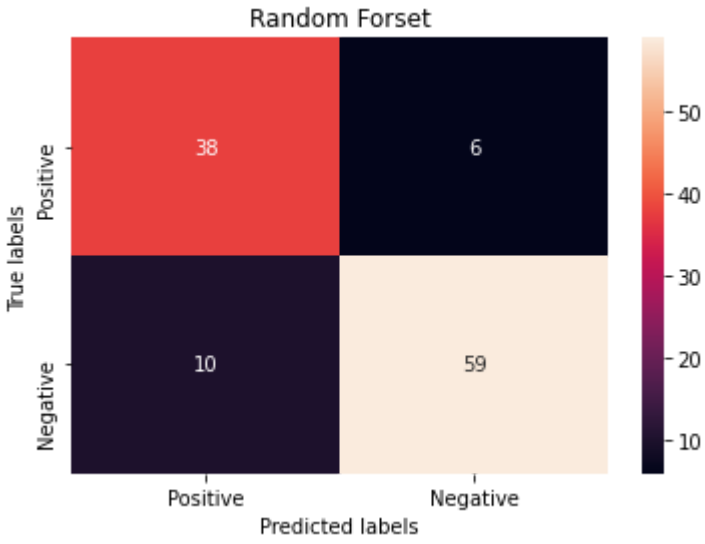
```
print("AUC score is: " + str("{0:.2f}".format(100 * metrics.roc_auc_score(y_test, y_pred
print("F1 score is: " + str("{0:.2f}".format(100 * metrics.f1_score(y_test, y_pred))) +
print("Loss is: " + str("{0:.2f}".format(metrics.log_loss(y_test, y_pred))))
print("Accuracy is: " + str("{0:.2f}".format(100 * metrics.accuracy_score(y_test, y_pred
```



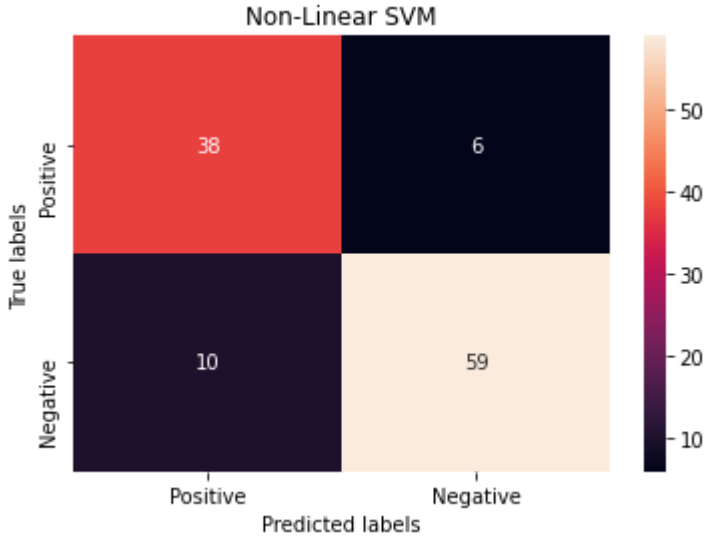

```
AUC score is: 85.94%
F1 score is: 88.06%
Loss is: 4.89
Accuracy is: 85.84%
```




```
AUC score is: 85.94%
F1 score is: 88.06%
Loss is: 4.89
Accuracy is: 85.84%
```




```
AUC score is: 85.94%
F1 score is: 88.06%
Loss is: 4.89
Accuracy is: 85.84%
```




```
AUC score is: 85.94%
F1 score is: 88.06%
Loss is: 4.89
Accuracy is: 85.84%
```

As we can clearly see, the models which trained on the reduced dimensionality preform better compared to the models which trained on the best two features from section 6