Implementação de Técnicas de Data Mining e Machine Learning em Bancos de Dados da saúde

- Matheus Fuhrmann Stigger
- Rodrigo Blanke Lambrecht

Dataset escolhido: Trata-se da coleta de semen de 100 voluntários com intuito de identificar a normalidade (ou não) da fertilidade dos mesmos, disponível no link https://archive.ics.uci.edu/ml/datasets/Fertility (https://archive.ics.uci.edu/ml/datasets/Fertility)

A concentração de espermatozóide encontrado no esperma está relacionada à fatores demográficos, ambientais, saúde do indivíduo e hábitos diários. Os dados disponibilizados encontram-se categorizados e disttribuídos nas seguintes colunas:

- season: Estação do ano em que a coleta foi realizada: 1) inverno, 2) primavera, 3) verão, 4) outono. (-1, -0.33, 0.33, 1)
- age: Idade do doador: 18-36 (0, 1);
- childdiseases: Doenças quando criança: 1) sim, 2) não. (0, 1);
- accident: Acidentes ou traumas sérios: 1) sim, 2) não. (0, 1) ;
- surgintv: Intervenção Surirgica: 1) sim, 2) não. (0, 1);
- highfevers: Febre alta no último ano: 1) menos de 3 meses atrás, 2) mais de 3 meses atrás, 3) não. (-1, 0, 1);
- freqalcohol: Consumo de álcool: 1) várias vezes ao dia, 2) todos os dias, 3) várias vezes por semana, 4) uma vez por semana, 5) quase nunca ou nunca (0.2, 0.4, 0.6, 0.8, 1);
- smoking: Hábito de fumar 1) nunca, 2) ocasionalmente 3) diário. (-1, 0, 1);
- nhsspde: Número de horas passadas sentadas por dia 0-16 (0, 1);
- outcome: Diagnóstico: Normal (N), Alterado (O).

In [93]:

```
import matplotlib.pyplot as plt
%matplotlib inline
import warnings

warnings.filterwarnings("ignore", message="numpy.dtype size changed")
warnings.filterwarnings("ignore", message="numpy.ufunc size changed")
```

Como mencionado em aula: esse trecho de código vai aparecer em todos os notebooks para suprimir warnings chatos de versão numpy

In [70]:

```
import pandas as pd
dataset_original = pd.read_csv('fertilidade_edit.csv', sep = ',')
print '%d amostras e %d colunas' % dataset_original.shape
#for col in dataset_original.columns:
#    print 'Primeiras 2 linhas da coluna %s:'%col
#    print dataset_original.loc[0:1,col]
```

```
100 amostras e 10 colunas
```

Não há necessidade de substituir valores do dataset pois o próprio informa que não há erros nem valores faltantes.

Alteração da coluna 'outcome': N (normal) substituído por '0' O (alterado) substituído por '1'

In [71]:

```
import numpy as np
for col in ['outcome']:
    dataset_original[col] = dataset_original[col].replace('N',0)
    dataset_original[col] = dataset_original[col].replace('O',1)

#for col in dataset_original.columns:
print dataset_original.loc[:,]
```

```
    season   age  childdiseases  accident  surgintv  highfevers  freqalcoh
ol  \
0    -0.33  0.69              0         1         1           0
0.8
1    -0.33  0.94              1         0         1           0
0.8
2    -0.33  0.50              1         0         0           0
1.0
3    -0.33  0.75              0         1         1           0
1.0
4    -0.33  0.67              1         1         0           0
0.8
5    -0.33  0.67              1         0         1           0
0.8
6    -0.33  0.67              0         0         0          -1
0.8
7    -0.33  1.00              1         1         1           0
0.6
8     1.00  0.64              0         0         1           0
```

Não há colunas com valores nulos

In [72]:

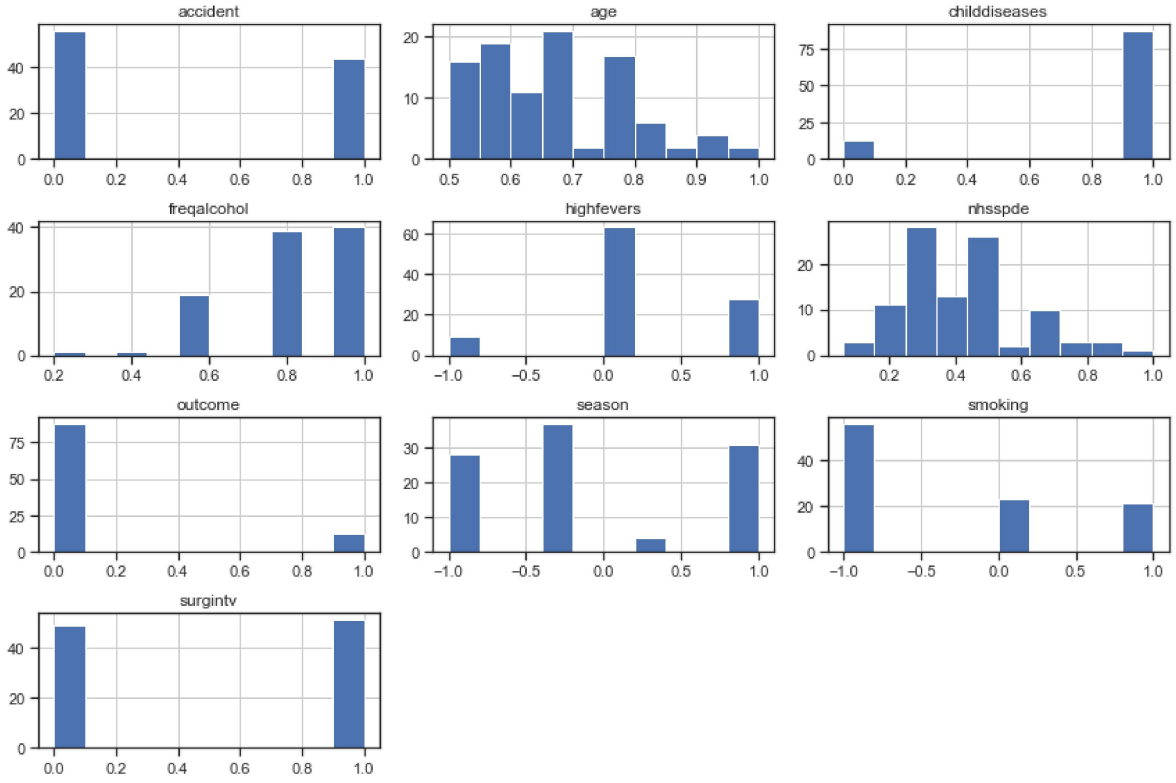
```
dataset_original.isnull().any()
```

Out[72]:

```
season           False
age              False
childdiseases    False
accident         False
surgintv         False
highfevers       False
freqalcohol      False
smoking          False
nhsspde          False
outcome          False
dtype: bool
```

In [73]:

```
fig,ax = plt.subplots(figsize = (12,8))
dataset_original.hist(ax = ax)


plt.tight_layout()
plt.show();
```



Sem necessidade de detecção de outliers pois os dados fornecidos já estão categorizados

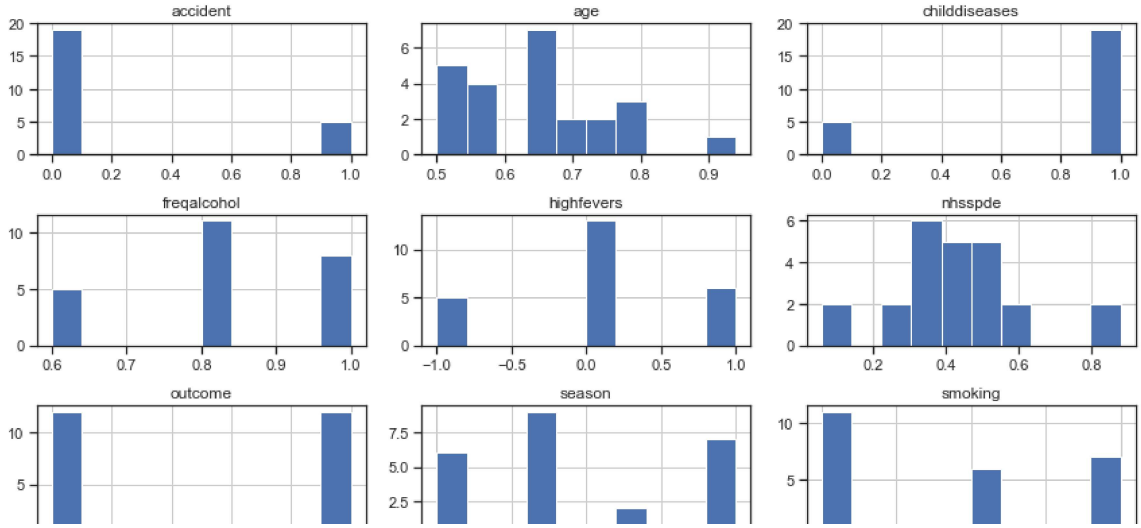
Utilizando under sampling para balancear as amostras. Xr array atributos | yr array outcome

Para utilização dos métodos imblearn é necessário instalar pack 'conda install -c glemaitre imbalanced-learn' através do anaconda prompt

In [74]:

```
count = np.unique(dataset_original.outcome, return_counts = True)
print 'Distribuição da classe sem under sampling %d %d' % (count[1][0], count[1][1])

from imblearn.under_sampling import RandomUnderSampler
X_under = dataset_original.loc[:, dataset_original.columns != 'outcome']
y_under = dataset_original.outcome

rus_under = RandomUnderSampler()
Xr_under, yr_under = rus_under.fit_sample(X_under,y_under)
count_under = np.unique(yr_under, return_counts = True)
print 'Distribuição da classe com under sampling %d %d' % (count_under[1][0], count_under[1

dataset_balanceado_under = pd.DataFrame(Xr_under, columns = dataset_original.columns[:-1])
dataset_balanceado_under["outcome"] = yr_under

fig,ax = plt.subplots(figsize = (12,8))
dataset_balanceado_under.hist(ax = ax)
plt.tight_layout()
plt.show();
```

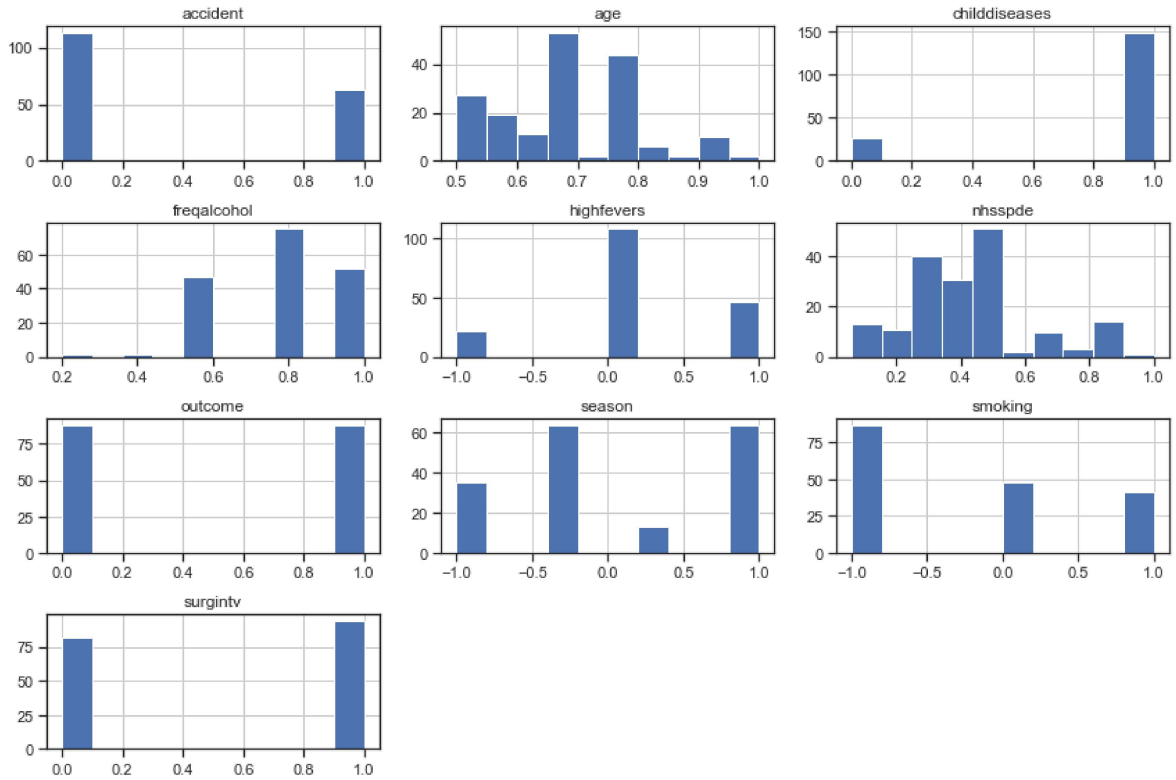
```
Distribuição da classe sem under sampling 88 12
Distribuição da classe com under sampling 12 12
```



In [75]:

```
count = np.unique(dataset_original.outcome, return_counts = True)
print 'Distribuição da classe sem over sampling %d %d' % (count[1][0], count[1][1])

from imblearn.over_sampling import RandomOverSampler
X_over = dataset_original.loc[:, dataset_original.columns != 'outcome']
y_over = dataset_original.outcome

ros_over = RandomOverSampler()
Xr_over, yr_over = ros_over.fit_sample(X_over,y_over)
count_over = np.unique(yr_over, return_counts = True)
print 'Distribuição da classe com over sampling %d %d' % (count_over[1][0], count_over[1][1

dataset_balanceado_over = pd.DataFrame(Xr_over, columns = dataset_original.columns[:-1])
dataset_balanceado_over["outcome"] = yr_over

fig,ax = plt.subplots(figsize = (12,8))
dataset_balanceado_over.hist(ax = ax)
plt.tight_layout()
plt.show();
```

```
Distribuição da classe sem over sampling 88 12
Distribuição da classe com over sampling 88 88
```



Denoising > Sem necessidade

Verificação da normalidade da distribuição dos atributos

In [76]:

```
from scipy.stats import shapiro
from scipy.stats import normaltest

for col in dataset_balanceado_over.columns[:-1]:
    stat, p = normaltest(dataset_balanceado_over[col])
    #print('Statistics=%.3f, p=%.3f' % (stat, p))
    alpha = 0.05
    if p > alpha: # H0: a distribuição é normal
        print 'parece Gaussiana %s - (falha ao rejeitar H0)' % (col)
    else:
        print 'não parece Gaussiana - %s (rejeita H0)' % (col)
```

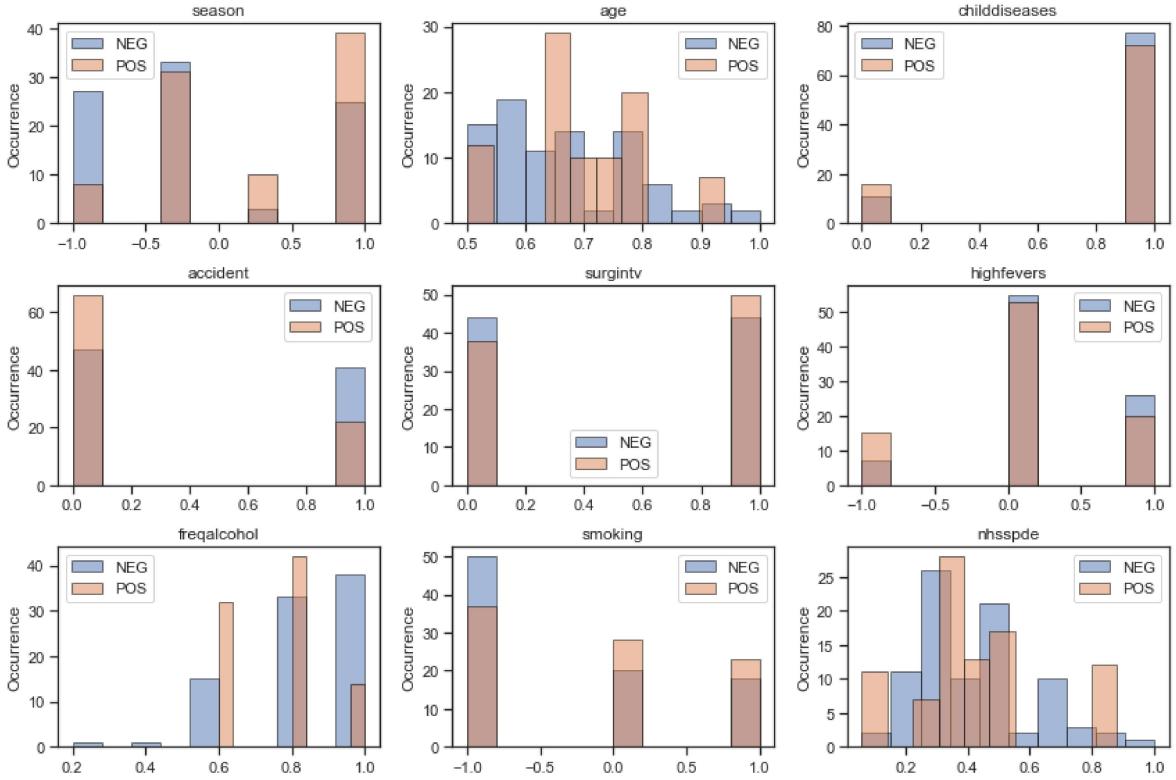
```
parece Gaussiana season - (falha ao rejeitar H0)
parece Gaussiana age - (falha ao rejeitar H0)
não parece Gaussiana - childdiseases (rejeita H0)
não parece Gaussiana - accident (rejeita H0)
parece Gaussiana surgintv - (falha ao rejeitar H0)
parece Gaussiana highfevers - (falha ao rejeitar H0)
parece Gaussiana freqalcohol - (falha ao rejeitar H0)
não parece Gaussiana - smoking (rejeita H0)
não parece Gaussiana - nhsspde (rejeita H0)
```

In [77]:

```
import matplotlib.pyplot as plt
from scipy.stats import anderson

fig,axes = plt.subplots(3,3,figsize = (12,8))

i = j = 0
for idx, col in enumerate(dataset_balanceado_over.columns[:-1]):
    # Grupo 0: Casos NEGATIVOS (NEG)
    tmp0 = dataset_balanceado_over.loc[dataset_balanceado_over.outcome == 0, col]
#     tmp0 = dataset_balanceado_over.loc[, col]
    # Grupo 1: Casos POSITIVOS (POS)
    tmp1 = dataset_balanceado_over.loc[dataset_balanceado_over.outcome == 1, col]
#     tmp1 = dataset_balanceado_over.loc[, col]

    stat, p = normaltest(tmp0)
    alpha = 0.05
    if p > alpha:
        print 'Distribuição %s parece Gaussiana (falha ao rejeitar H0), group NEG' % (col)
    else:
        print 'Distribuição %s não parece Gaussiana (rejeita H0), group NEG' % (col)

    stat, p = normaltest(tmp1)
    alpha = 0.05
    if p > alpha:
        print 'Distribuição %s parece Gaussiana (falha ao rejeitar H0), group POS' % (col)
    else:
        print 'Distribuição %s não parece Gaussiana (rejeita H0), group POS' % (col)



    i = idx / 3
    j = idx % 3

    axes[i][j].hist(tmp0, label ='NEG', alpha = 0.5, lw = 1,edgecolor='black')
    axes[i][j].hist(tmp1, label = 'POS', alpha = 0.5, lw = 1,edgecolor='black')
    axes[i][j].set_title(col)
    axes[i][j].set_ylabel('Occurrence')
    axes[i][j].legend()




plt.tight_layout()
plt.show();
```

```
Distribuição season não parece Gaussiana (rejeita H0), group NEG
Distribuição season não parece Gaussiana (rejeita H0), group POS
Distribuição age não parece Gaussiana (rejeita H0), group NEG
Distribuição age parece Gaussiana (falha ao rejeitar H0), group POS
Distribuição childdiseases não parece Gaussiana (rejeita H0), group NEG
Distribuição childdiseases não parece Gaussiana (rejeita H0), group POS
Distribuição accident parece Gaussiana (falha ao rejeitar H0), group NEG
Distribuição accident não parece Gaussiana (rejeita H0), group POS
Distribuição surgintv parece Gaussiana (falha ao rejeitar H0), group NEG
Distribuição surgintv parece Gaussiana (falha ao rejeitar H0), group POS
Distribuição highfevers parece Gaussiana (falha ao rejeitar H0), group NEG
Distribuição highfevers parece Gaussiana (falha ao rejeitar H0), group POS
```

```
Distribuição freqalcohol não parece Gaussiana (rejeita H0), group NEG
Distribuição freqalcohol não parece Gaussiana (rejeita H0), group POS
Distribuição smoking não parece Gaussiana (rejeita H0), group NEG
Distribuição smoking não parece Gaussiana (rejeita H0), group POS
Distribuição nhsspde não parece Gaussiana (rejeita H0), group NEG
Distribuição nhsspde parece Gaussiana (falha ao rejeitar H0), group POS
```



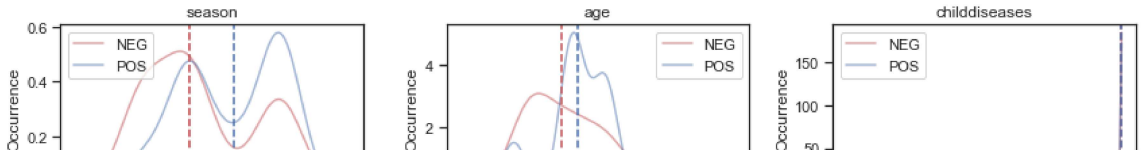
Relevâncias:

- Idade (Age) - maior probabilidade de ocorrência na faixa de 0,65 a 0,8
- Consumo de álcool (freqalcohol) - 0 a 0,6 não ocorre e 0,6 a 1 aumenta a probabilidade
- tempo sentado (nhsspde) - 0,2 a 0,55 aumenta probabilidade

Plotar PDF para facilitar a visualição

In [78]:

```
import seaborn as sns

fig,axes = plt.subplots(3,3,figsize = (12,8))

i = j = 0
for idx, col in enumerate(dataset_balanceado_over.columns[:-1]):
    tmp0 = dataset_balanceado_over.loc[dataset_balanceado_over.outcome == 0, col]
    tmp1 = dataset_balanceado_over.loc[dataset_balanceado_over.outcome == 1, col]

    i = idx / 3
    j = idx % 3

    sns.kdeplot(tmp0, ax = axes[i][j], label ='NEG', alpha = 0.5,color='r')
    sns.kdeplot(tmp1,ax = axes[i][j], label ='POS', alpha = 0.5,color='b')

    # cálculo das medianas e plotagem com a função axvline(x = mediana)
    median0 = tmp0.median()
    median1 = tmp1.median()

    axes[i][j].axvline(x=median0,color='r', linestyle='--')
    axes[i][j].axvline(x=median1,color='b', linestyle='--')

    axes[i][j].set_title(col)
    axes[i][j].set_ylabel('Occurrence')
    axes[i][j].legend()

    # vamos adicionar uma caixinha com a distância entre as medianas
    # vamor normalizar para interpretar essa distância independente da escala

    diff_med = abs(median0 - median1)
    min_val = dataset_balanceado_over[col].min()
    max_val = dataset_balanceado_over[col].max()
    diff_med_norm = (diff_med)/(max_val - min_val)

    axes[i][j].set_xlabel( r'$\Delta$Mediana$_{\mathrm{NORM}}$ = %.2f' % (diff_med_norm), f



plt.tight_layout()
plt.show();
```

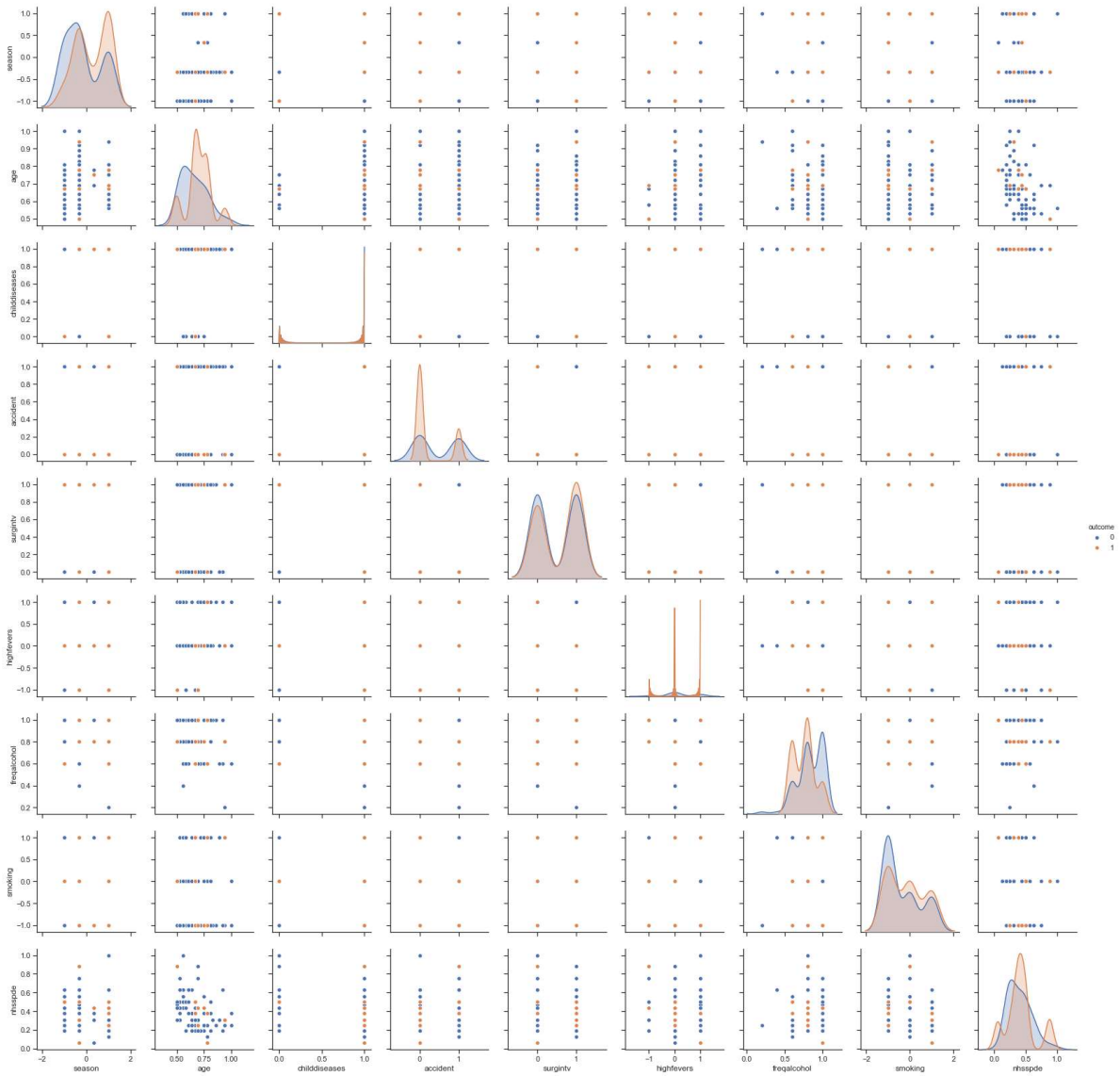


In [79]:

```
sns.set(style="ticks")
attributes = dataset_balanceado_over.columns.drop("outcome")
sns.pairplot(dataset_balanceado_over, hue="outcome", vars=attributes)
```

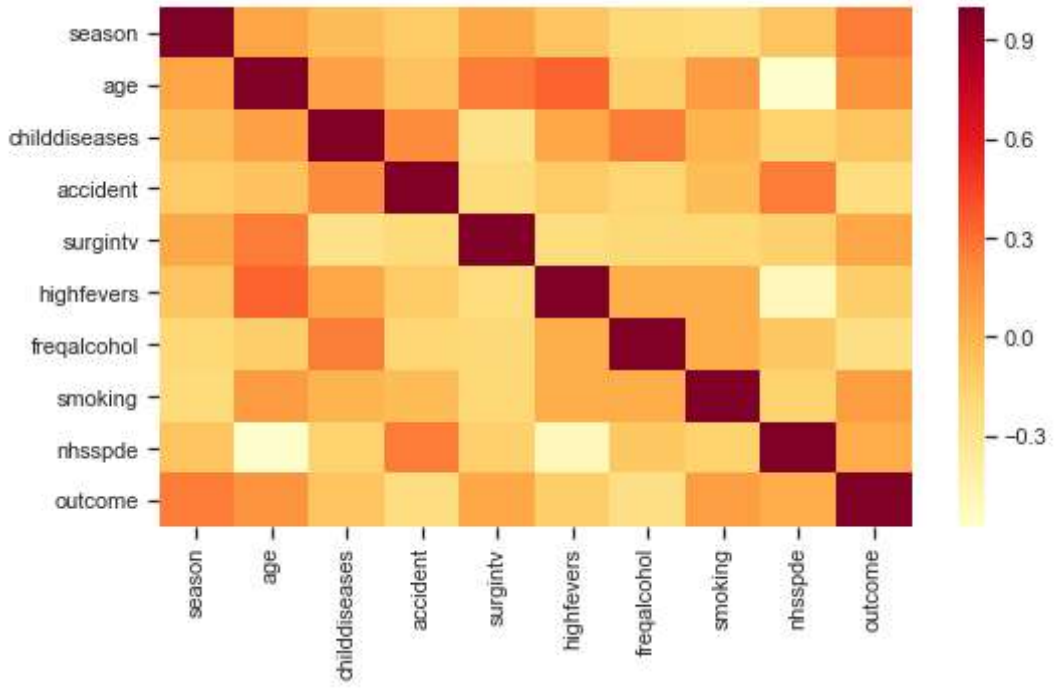
Out[79]:

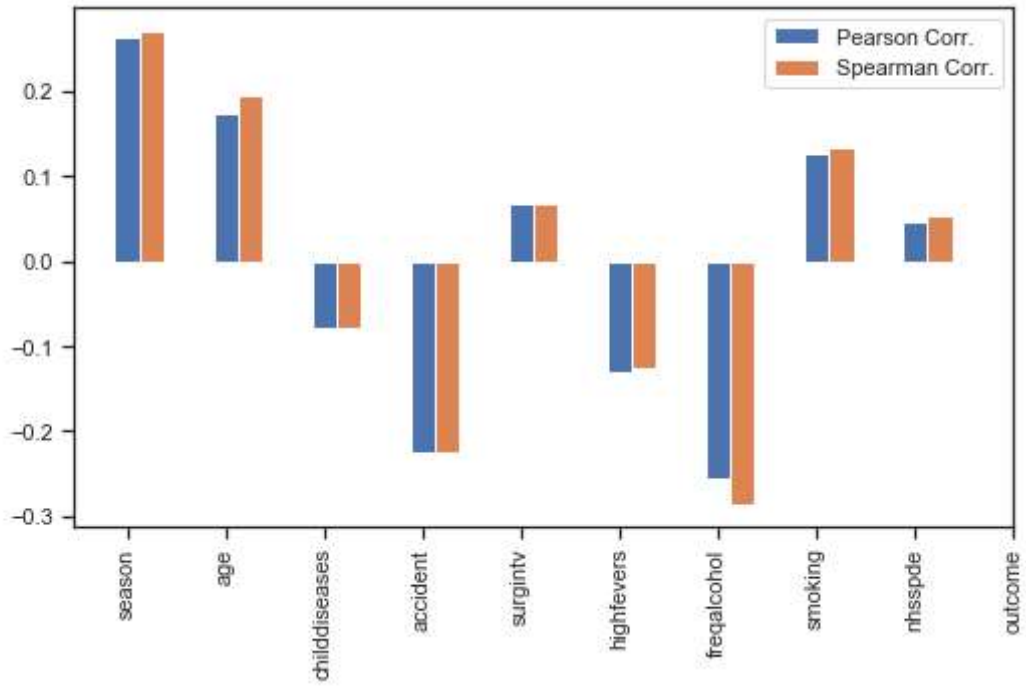
```
<seaborn.axisgrid.PairGrid at 0x23dbf358>
```



In [80]:

```
import seaborn as sns

fig, axes = plt.subplots(2,1,figsize=(8,10))

# plotando as correlações entre atributos usando Pearson
sns.heatmap(dataset_balanceado_over.corr(), cmap = 'YlOrRd', ax = axes[0])


corr_pearson_vet = []
corr_spearman_vet = []
atributos = dataset_balanceado_over.columns.tolist()
atributos

# para calcular Spearman, vamos ter que percorrer todas as colunas e fazer caso a caso
for col in dataset_balanceado_over.columns:
    if col == 'outcome': continue


    corr_pearson = dataset_balanceado_over[col].corr(dataset_balanceado_over['outcome'], me
    corr_spearman = dataset_balanceado_over[col].corr(dataset_balanceado_over['outcome'], m

    corr_pearson_vet.append(corr_pearson)
    corr_spearman_vet.append(corr_spearman)


# aqui vamos plotar as barras com as correlações de Pearson e Spearman
axes[1].bar(np.arange(len(corr_pearson_vet)),corr_pearson_vet, width=0.25, label = "Pearsor
axes[1].bar(np.arange(len(corr_pearson_vet))+0.25,corr_spearman_vet, width=0.25, label = "S

# aqui vamos renomear os rótulos do eixo X para colocar os nomes dos nossos atributos
axes[1].set_xticks(range(len(atributos)))
axes[1].set_xticklabels(atributos, rotation=90)

axes[1].legend()

plt.tight_layout()
plt.show()
```

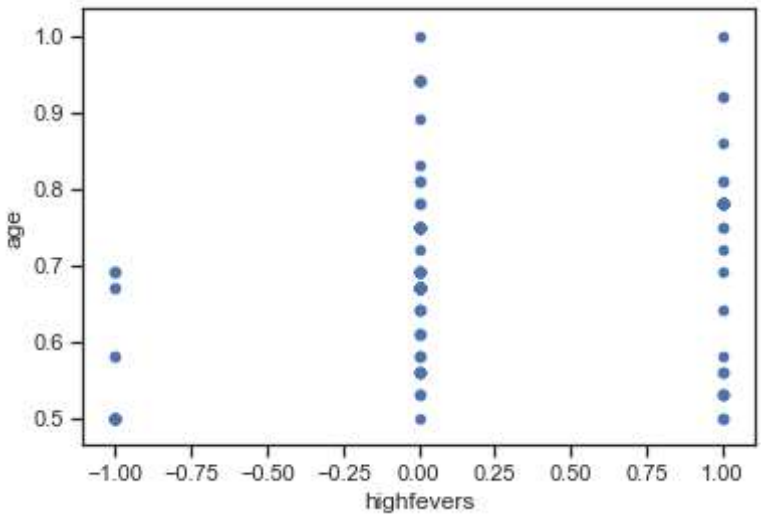




Maior relação Idade (Age) com Febre Alta no último ano (highfeavers)

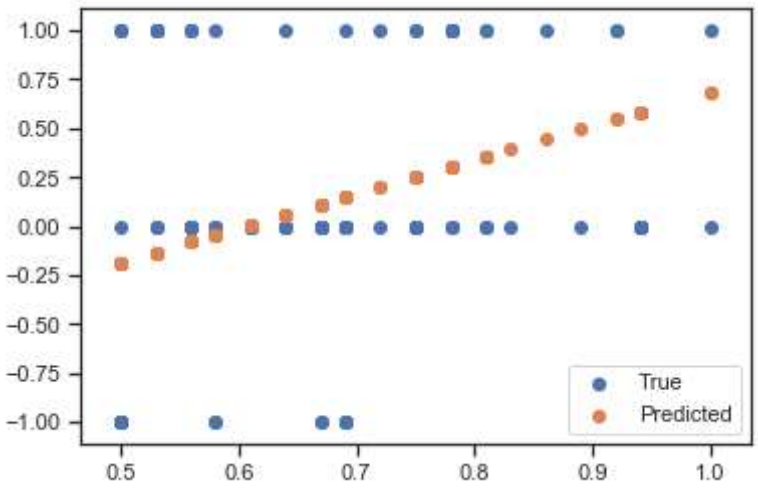
In [81]:

```
dataset_balanceado_over.plot.scatter('highfevers','age')
plt.show()
```



In [82]:

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

X = dataset_balanceado_over['age'].values.reshape(-1,1)
y =  dataset_balanceado_over['highfevers']

# aqui treinamis um modelo de regressao linear, pois SkinThickness e BMI sao numericas
# o metodo fit(X,y) faz todo o trabalho de encontrar a reta mais adequada
rl = LinearRegression().fit(X,y)

# o metodo predict recebe os atributos como entrada e utiliza o modelo treinado
# para computar os desfechos preditos
y_pred = rl.predict(X)


# a funcao scatter imprime um grafico de dispersao X1,X2 na tela
# vamos plotar o BMI real e o predito
plt.scatter(X, y, label = 'True')
plt.scatter(X, y_pred, label = 'Predicted')

plt.legend()

plt.show()

# uma forma comum de avaliar a eficiência de um modelo de regressão linear é pelo
# Erro Médio Quadrático (Mean Squared Error - MSE)
# MSE(y_true, y_pred) = sum ((y_true - y_pred)^2)/|y_true|

mse = mean_squared_error(y, y_pred)
print 'coeficiente de regressão', rl.coef_
print 'coeficiente linear', rl.intercept_

print 'MSE', mse
```




```
coeficiente de regressão [1.74631219]
coeficiente linear -1.0585703260682178
MSE 0.32522478585899955
```

Coeficiente diferente de zero (1.53) portanto tem relação significativa entre os dois atributos MSE baixo

(aproximadamente 30%)

In [83]:

```
X = dataset_balanceado_over[['age', 'freqalcohol']]
y =  dataset_balanceado_over['highfevers']

rl = LinearRegression().fit(X,y)
y_pred = rl.predict(X)


# uma forma comum de avaliar a eficiência de um modelo de regressão linear é pelo
# Erro Médio Quadrático (Mean Squared Error - MSE)
# MSE(y_true, y_pred) = sum ((y_true - y_pred)^2)/|y_true|

mse = mean_squared_error(y, y_pred)
print 'coeficiente de regressão: \n%.4f (Age) \n%.4f (Alcool)' % (rl.coef_[0],rl.coef_[1])
print 'coeficiente linear', rl.intercept_

print 'MSE', mse
```

```
coeficiente de regressão:
1.8041 (Age)
0.3151 (Alcool)
coeficiente linear -1.3502046752089
MSE 0.32274333627012397
```

Fazendo comparação com o consumo de álcool comprava-se que a febre continua tendo uma relação mais significativa com a idade do que com o consumo de álcool

Agora utilizando método "Generalized Linear Model (GLM)":

In [84]:

```
# statsmodel permite gerar modelos de regressão com resultados estatísticos
import statsmodels.api as sm
from scipy import stats


X = dataset_balanceado_over.loc[:,['age','freqalcohol']]
y = dataset_balanceado_over['highfevers']

print X.shape, y.shape

X2 = sm.add_constant(X)
est = sm.GLM(y, X2)
est2 = est.fit()
print(est2.summary())
```

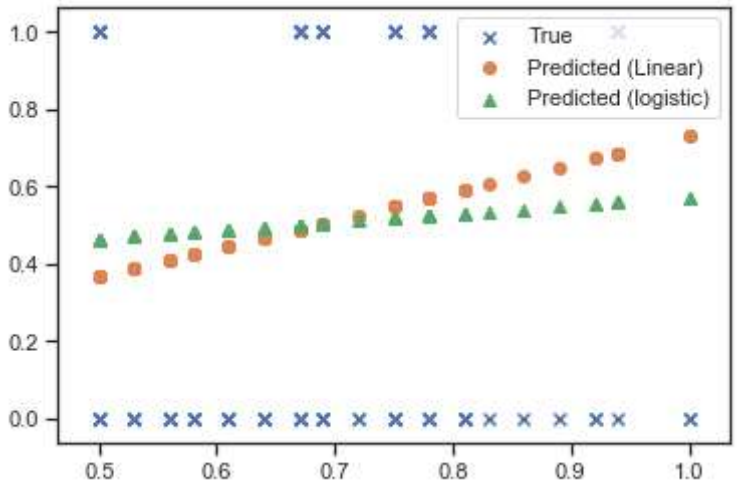
```
(176, 2) (176L,)
                 Generalized Linear Model Regression Results

==============================================================================
==
Dep. Variable:             highfevers   No. Observations:                  1
76
Model:                            GLM   Df Residuals:                      1
73
Model Family:                Gaussian   Df Model:
2
Link Function:               identity   Scale:                         0.328
34
Method:                          IRLS   Log-Likelihood:                -150.
21
Date:                Wed, 12 Dec 2018   Deviance:                       56.8
03
Time:                        22:03:46   Pearson chi2:                    5
6.8
No. Iterations:                     3   Covariance Type:             nonrobu
st
==============================================================================
===
                 coef    std err          z      P>|z|      [0.025      0.9
75]
------------------------------------------------------------------------------
---
const          -1.3502      0.358     -3.768      0.000      -2.053      -0.
648
age             1.8041      0.369      4.888      0.000       1.081       2.
527
freqalcohol     0.3151      0.273      1.153      0.249      -0.220       0.
851
==============================================================================
===
```

O p-value sendo <0.05 indica que os dados tem significância estatística que ocorre em Idade (age) e não ocorre com o consumo de álcool (freqalcohol)

Agora testando regressão logística:

In [85]:

```
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.metrics import mean_squared_error

X = dataset_balanceado_over['age'].values.reshape(-1,1)
y =  dataset_balanceado_over['outcome']

# Vamos criar dois preditores: um logistico e um linear para comparar os resultados
rl = LinearRegression().fit(X,y)
rlog = LogisticRegression().fit(X,y)


# o metodo predict recebe os atributos como entrada e utiliza o modelo treinado
# para computar os desfechos preditos
y_pred = rl.predict(X)

# O método predict_proba(X) retorna a probabilidade de cada desfecho
# ao pegarmos a coluna 1, estamos selecionando a probabilidade de desfecho positivo
y_pred_log = rlog.predict_proba(X)[:,1]


# a funcao scatter imprime um grafico de dispersao X1,X2 na tela
# vamos plotar o BMI real e o predito
plt.scatter(X, y, label = 'True', marker = 'x')
plt.scatter(X, y_pred, label = 'Predicted (Linear)', marker = 'o')
plt.scatter(X, y_pred_log, label = 'Predicted (logistic)', marker = '^')

plt.legend()

plt.show()

# uma forma comum de avaliar a eficiência de um modelo de regressão linear é pelo
# Erro Médio Quadrático (Mean Squared Error - MSE)
# MSE(y_true, y_pred) = sum ((y_true - y_pred)^2)/|y_true|

mse_lin = mean_squared_error(y, y_pred)
mse_log = mean_squared_error(y, y_pred_log)

print 'coeficiente de regressão linear', rl.coef_
print 'coeficiente de regressão logística', rlog.coef_


print 'MSE linear', mse_lin

print 'MSE logístico', mse_log
```




```
coeficiente de regressão linear [0.73107038]
coeficiente de regressão logística [[0.86715922]]
MSE linear 0.2425439129343131
MSE logístico 0.246242389239546
```

In [86]:

```
X = dataset_balanceado_over.loc[:,['age','highfevers', 'season']]
y = dataset_balanceado_over['outcome']

print X.shape, y.shape

X2 = sm.add_constant(X)
est = sm.Logit(y, X2)
est2 = est.fit()

print(est2.summary())
print 'Odds Ratio:'
print np.exp(est2.params)


# por padrao stats model retorna a probabilidade (equivalente ao predict_proba())
y_pred_proba = est2.predict(X2)
# vamo converter todas as saidas acima de 0.5 para 1 e tornar nosso desfecho binario
y_pred_01 = (y_pred > 0.5).astype(int)


# a taxa de acerto (accuracy) eh a media de respostas corretas
accuracy = (y_pred_01 == y).mean()
print 'Taxa de acertos: %.2f%%' % (accuracy*100)
```

```
(176, 3) (176L,)
Optimization terminated successfully.
         Current function value: 0.628655
         Iterations 5
                           Logit Regression Results

==============================================================================
==
Dep. Variable:                outcome   No. Observations:                   1
76
Model:                          Logit   Df Residuals:                       1
72
Method:                           MLE   Df Model:
3
Date:                Wed, 12 Dec 2018   Pseudo R-squ.:                  0.093
04
Time:                        22:03:53   Log-Likelihood:                -110.
64
converged:                       True   LL-Null:                       -121.
99
                                        LLR p-value:                 4.661e-
05
==============================================================================
==
                 coef    std err          z      P>|z|      [0.025      0.97
5]
------------------------------------------------------------------------------
--
const         -2.8645      1.069     -2.679      0.007      -4.960      -0.7
69
age            4.2414      1.563      2.714      0.007       1.178       7.3
04
```

```
highfevers    -0.7258      0.301     -2.414      0.016      -1.315      -0.1
37
season         0.6409      0.211      3.037      0.002       0.227       1.0
55
==============================================================================
==
Odds Ratio:
const          0.057014
age           69.503513
highfevers     0.483942
season         1.898179
dtype: float64
Taxa de acertos: 57.39%
```

Os 3 atributos escolhidos mantém um p-value < 0.05 e a idade tem um odd ratio de 91x , portanto a cada incremento de 1 na idade temos um aumento de 91 vezes na chance de ter problemas de fertilidade.

In [87]:

```
from sklearn.metrics import confusion_matrix

X = dataset_balanceado_over.loc[:,['age','highfevers', 'season']]
y = dataset_balanceado_over['outcome']

rlog = LogisticRegression().fit(X,y)
y_pred_log = rlog.predict(X)
print (y_pred_log == 0).sum()
cm = confusion_matrix(y, y_pred_log)

print 'Matriz de Confusao'
print cm

print 'TPR', cm[1][1]/float(cm[1][0] + cm[1][1])
print 'TNR', cm[0][0]/float(cm[0][0] + cm[0][1])
```

```
74
Matriz de Confusao
[[54 34]
 [20 68]]
TPR 0.7727272727272727
TNR 0.6136363636363636
```

As taxas de verdadeiro/positivo falso/positivos encontradas foram consideradas aceitáveis.

In [88]:

```
from statsmodels.stats.mediation import Mediation
import statsmodels.api as sm


independent = "age"
mediator = "season"
outcome = "outcome"


X = dataset_balanceado_over[independent]
X2 = sm.add_constant(X)
M = dataset_balanceado_over[mediator]

mediator_model = sm.GLM(M, X2)

X = dataset_balanceado_over.loc[:,[independent, mediator]]
X2 = sm.add_constant(X)
Y = dataset_balanceado_over[outcome]

outcome_model = sm.Logit(Y, X2)

#Mediation(outcome_model, mediator_model, exposure, mediator=None,...)

med = Mediation(outcome_model, mediator_model, independent, mediator=mediator).fit()

print(med.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.646195
         Iterations 5
                         Estimate  Lower CI bound  Upper CI bound  P-value
ACME (control)           0.046786       -0.025744        0.165781    0.218
ACME (treated)           0.079974       -0.045543        0.241477    0.220
ADE (control)            0.482297        0.028929        0.786664    0.028
ADE (treated)            0.515485        0.029041        0.816875    0.028
Total effect             0.562271        0.108970        0.834251    0.014
Prop. mediated (control) 0.061053       -0.065918        0.578647    0.228
Prop. mediated (treated) 0.130430       -0.128653        0.612511    0.230
ACME (average)           0.063380       -0.034946        0.199377    0.218
ADE (average)            0.498891        0.028985        0.797912    0.028
Prop. mediated (average) 0.101346       -0.086536        0.599609    0.228
```

Valores p-valeu de ACME maiores que 0.05 indicam a não relação de mediação com a idade (age).

Utilizando a métrica GINI index para avaliar o atributo de maior importância novamente:

In [89]:

```
from sklearn.tree import DecisionTreeClassifier

X = dataset_balanceado_over.loc[:,dataset_balanceado_over.columns != 'outcome']

y = dataset_balanceado_over['outcome']
clf = DecisionTreeClassifier().fit(X,y)
gini = clf.feature_importances_

for feat, gini_val in zip(X.columns, gini):
    print '%s\t%.3f' % (feat, gini_val)
```

```
season  0.099
age     0.180
childdiseases   0.001
accident        0.169
surgintv        0.089
highfevers      0.102
freqalcohol     0.012
smoking 0.132
nhsspde 0.216
```

Percebe-se novamente a importância do atributo idade (age) para o outcome do nosso modelo preditivo

Refinamento:

In [90]:

```
from sklearn.model_selection import train_test_split

X = dataset_balanceado_over.loc[:,dataset_balanceado_over.columns != 'outcome']
y =  dataset_balanceado_over['outcome']

# o metodo train_test_split retorna Xtreino, Xteste, Ytreino, Yteste
# o parametro test_size = 0.3 significa que vamos usar 30% para teste
# o parametro shuffle vai embaralhar as amostras antes
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, shuffle = True)

print '# amostras de treino: ', X_train.shape[0]
print '# amostras de teste : ', X_test.shape[0]
```

```
# amostras de treino:  123
# amostras de teste :  53
```

In [91]:

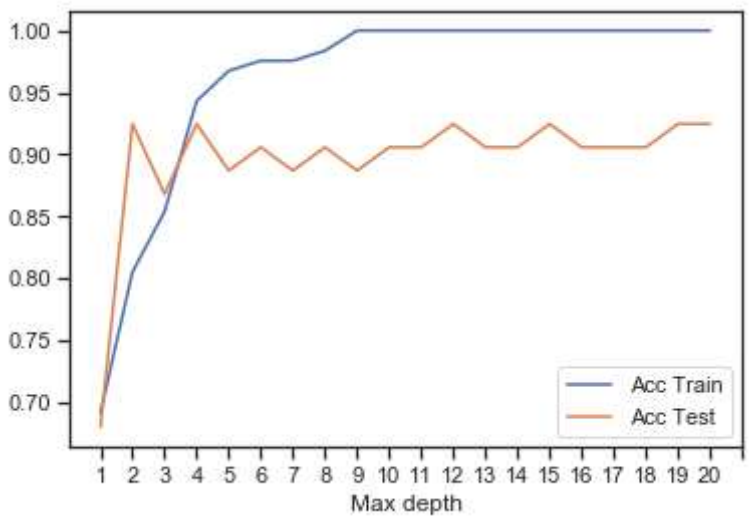
```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_validate

# vamos testar dois algoritmos bem diferentes: arvores e regressao log.
tree = DecisionTreeClassifier()
logreg = LogisticRegression()

# cross_validate faz a validacao cruzada para cv = N folds. Nesse caso vamos usar 5.
# o metodo suporta diferentes metricas, mas vamos usar accuracy
results_tree = cross_validate(tree, X_train, y_train, cv = 5, scoring ='accuracy')
results_reg = cross_validate(logreg, X_train, y_train, cv = 5, scoring ='accuracy')

# o retorno do cross validate eh um dicionario, onde cada chave representa um tipo de resul
#  Vamos plotar aqui o 'test_score' que eh um vetor com  o resultado de accuracy do teste p
print 'Média CV para Arv. Decisao', results_tree['test_score'].mean()
print 'Média CV para Reg. Logistica', results_reg['test_score'].mean()


# agora vamos treinar nossos modelos usando todo o conjunto de treino e comparar com a
# accuracy obtida no conjunto de teste
tree = tree.fit(X_train,y_train)
y_pred1 = tree.predict(X_test)


svc = logreg.fit(X_train,y_train)
y_pred2 = logreg.predict(X_test)


print 'Acc. para conjunto de teste Arv. Decisao: %.2f%%' % ((y_pred1 == y_test).mean()*100)
print 'Acc. para conjunto de teste Reg. Logistica: %.2f%%'  % ((y_pred2 == y_test).mean()*1
```

```
Média CV para Arv. Decisao 0.8390000000000001
Média CV para Reg. Logistica 0.6733333333333333
Acc. para conjunto de teste Arv. Decisao: 96.23%
Acc. para conjunto de teste Reg. Logistica: 66.04%
```

Árvore de decisão se mostra mais eficiente do que reg. logistica

In [92]:

```
X = dataset_balanceado_over.loc[:,dataset_balanceado_over.columns != 'outcome']
y =  dataset_balanceado_over['outcome']

# separando 30% para teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, shuffle = True)

# vamos criar dois vetores que vao guardar as acuracias de treino e teste
# para cada arvore gerada (com diferentes valores de prof. maxima)
acc_train = []
acc_test = []
for maxd in range(1,21):
    tree = DecisionTreeClassifier(max_depth = maxd)
    tree = tree.fit(X_train, y_train)

    acc_train.append(  (tree.predict(X_train) == y_train).mean() )
    acc_test.append( (tree.predict(X_test) == y_test).mean() )


plt.plot(acc_train, label = 'Acc Train')
plt.plot(acc_test, label = 'Acc Test')
plt.legend()
plt.xticks(range(0,21), range(1,21))
plt.xlabel('Max depth')

plt.show()
```



Com base no gráfico acima podemos afirmar que não é necessário uma AD com profundida maior que 5 (Max depth)