```
== Goals ==

Create a model to predict churn and help churn reduction.
Try different tools provided by scikit-learn package, including PCA and PLS.

== Work flow ==

Data was extracted from the spreadsheet and samples were shuffled.
Data forms predictor and response matrices (X,Y) (test,train) with float values.
Three unique area codes were converted to numeric variable.
All these codes were from California, and state column was decided to be ignored.

Churn prediction problem can be solved as binary classification.
The classifier output can be a real value (continuous).
For example, it can be probability of churn.
In this case the boundary between classes is determined by a threshold value.
Result of the prediction can contain errors of two types:
1) false positive (churn predicted, actually no churn),
2) false negative (no churn predicted, actually was churn).
Taking different threshold values, we can reduce one, enlarging other.
So this value is determined by these error comparative costs.
Quality of classifier can be estimated
    by ROC curve (receiver operating characteristic) and AUC value.

Linear regression model was successfully tried.
PLS model behaved a slightly better than simple linear, but not notably.
AUC > 0.8 (blue ROC curves).

PCA was tried and score and loadings plots reviewed.
PCA is a way to reduce dimensionality of data --
    compose principal components from redundant variables (linear combination).
Score plots show samples in the space of some pair of principal components.
Loadings plots show how original variables affect pair of principal components.
One of score plots showed two clusters of samples.
The corresponding loadings plot showed connection with vmail* and night* stuff.
However clusters were not stable and can disappear with another training subset.

PCA mostly works better with scaled data and can be used in a PCR pipeline, like:
    scaling --> PCA --> linear regression.
Dimensionality reduces to about 13 components.
But the real power of PCR and PLS can be seen
    when number of variables is more than number of samples.
In our case this worked as good as simple linear regression.

Logistic regression and support vector machines were briefly tried,
but not successfully due to bad tuning or overall incompatibility with the task.
```

The best result was given by using decision tree classifier (green ROC curve).
Tree with maximal depth 5 seems to work well.
Making it bigger had not much improved the model or even made it overfitted.
Pipelining with PCA was not successful.

Decision tree is well observable (white box).
However it greatly changes appearance with training set reselection.
Variables, that was mostly mentioned in the tree:
CustServ Calls, Day Mins, Int’l Plan, VMail Message, Eve Mins.
Suggestions are that customers are likely to churn, having:
-- int’l plan,
-- talking much w/o vmail messages,
-- calling support (m. b. angry with problems) and not talking much.
These can happen due to customers personal unstable behaviour.
These can happen due to corresponding competitor’s proposal.

== More things to try ==

Clusters discovered by PCA can be then handled by separate models,
then accuracy may rise.

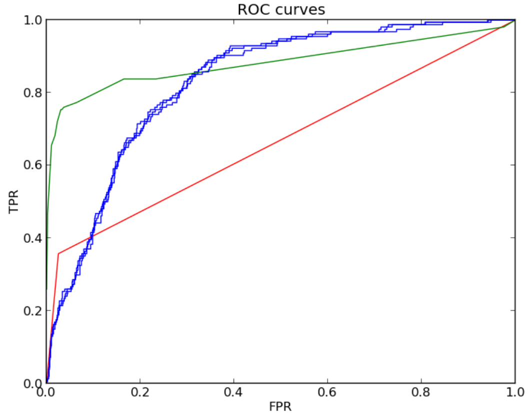
Variables have different nature.
It may be good to observe their distribution and preprocess them separately,
for example, applying logarithmic scale.

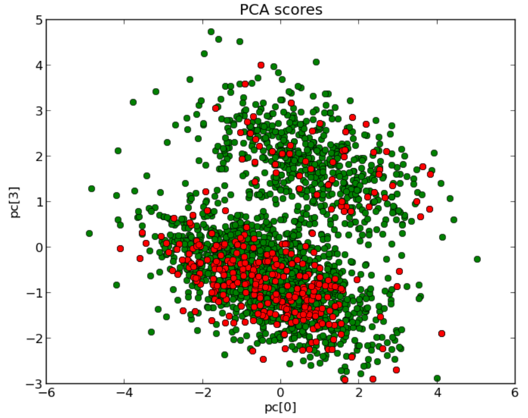
States and codes are not comparable as numbers.
It may be better to map each of them to
as many variables as there are different options, getting sparse matrix.
For example, we’ll have zeroes and ones in column California.

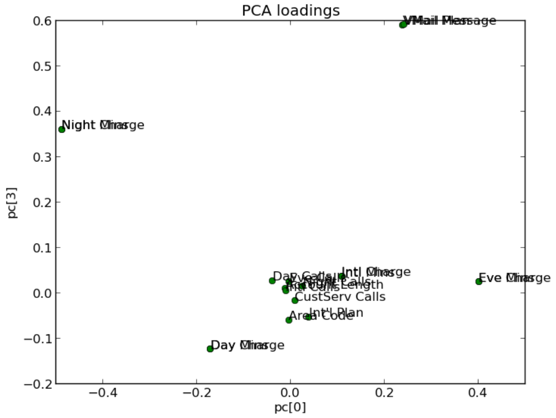
Model may work in pair.
Samples may be handled by decision tree,
but unclear ones forwarded to another model.

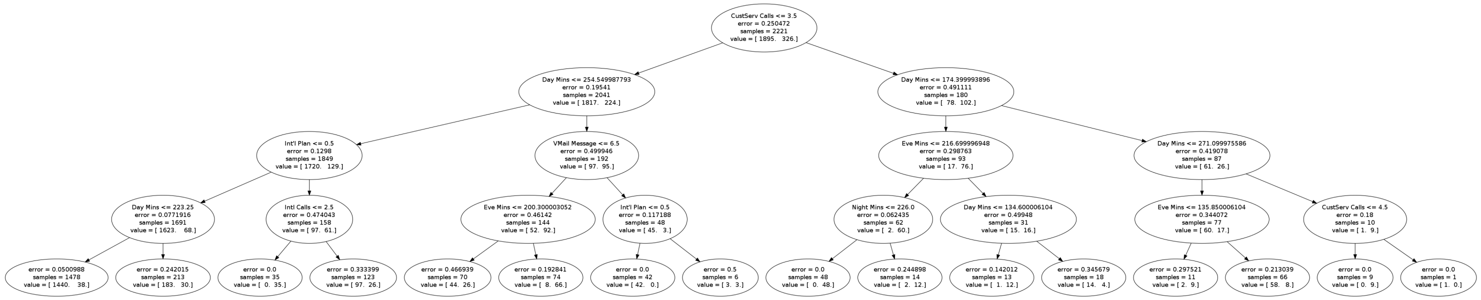
Bayesian networks and neural networks can be tried.

Decision tree options (like min_samples_leaf) can be tuned.


ROC curves
TPR
FPR
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0


PCA scores
pc[3]
pc[0]
5
4
3
2
1
0
−1
−2
−3
−6
−4
−2
0
2
4
6


PCA loadings
pc[3]
pc[0]
VMail Plan
VMail Message
Night Mins
Night Charge
Intl Mins
Intl Charge
Day Calls
Eve Calls
Night Calls
Account Length
Intl Calls
Eve Mins
Eve Charge
CustServ Calls
Int'l Plan
Area Code
Day Mins
Day Charge
0.6
0.5
0.4
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.4
−0.2
0.0
0.2
0.4


CustServ Calls <= 3.5
error = 0.250472
samples = 2221
value = [1895.   326.]
Day Mins <= 254.549987793
error = 0.19541
samples = 2041
value = [1817.   224.]
Day Mins <= 174.399993896
error = 0.491111
samples = 180
value = [78. 102.]
Int'l Plan <= 0.5
error = 0.1298
samples = 1849
value = [1720.   129.]
VMail Message <= 6.5
error = 0.499946
samples = 192
value = [97. 95.]
Eve Mins <= 216.699996948
error = 0.298763
samples = 93
value = [17. 76.]
Day Mins <= 271.099975586
error = 0.419078
samples = 87
value = [61. 26.]
Day Mins <= 223.25
error = 0.0771916
samples = 1691
value = [1623.   68.]
Intl Calls <= 2.5
error = 0.474043
samples = 158
value = [97. 61.]
Eve Mins <= 200.300003052
error = 0.46142
samples = 144
value = [52. 92.]
Int'l Plan <= 0.5
error = 0.117188
samples = 48
value = [45.   3.]
Night Mins <= 226.0
error = 0.062435
samples = 62
value = [2. 60.]
Day Mins <= 134.600006104
error = 0.49948
samples = 31
value = [15. 16.]
Eve Mins <= 135.850006104
error = 0.344072
samples = 77
value = [60. 17.]
CustServ Calls <= 4.5
error = 0.18
samples = 10
value = [1. 9.]
error = 0.0500988
samples = 1478
value = [1440.   38.]
error = 0.242015
samples = 213
value = [183.   30.]
error = 0.0
samples = 35
value = [0. 35.]
error = 0.333399
samples = 123
value = [97. 26.]
error = 0.466939
samples = 70
value = [44. 26.]
error = 0.192841
samples = 74
value = [8. 66.]
error = 0.0
samples = 42
value = [42.   0.]
error = 0.5
samples = 6
value = [3. 3.]
error = 0.0
samples = 48
value = [0. 48.]
error = 0.244898
samples = 14
value = [2. 12.]
error = 0.142012
samples = 13
value = [1. 12.]
error = 0.345679
samples = 18
value = [14.   4.]
error = 0.297521
samples = 11
value = [2. 9.]
error = 0.213039
samples = 66
value = [58.   8.]
error = 0.0
samples = 9
value = [0. 9.]
error = 0.0
samples = 1
value = [1. 0.]

See python sources at https://github.com/rewlad/ttu_andm

```

import glob
import csv
import numpy as np
import matplotlib.pyplot as plt
from sklearn import pipeline, preprocessing, decomposition, linear_model, pls, tree, metrics, utils

def load_rows():
    with open('Churn.csv','rb') as csvfile:
        reader = csv.reader(csvfile,delimiter=',',quotechar='"')
        rows = []
        for rn,row in enumerate(reader):
            if rn==0: head = row[0:22]
            else: rows.append(row[0:22])
        return rows, head

def rows_to_predictor_response(rows, head):
    codes = [row[19] for row in rows]
    codes_h = { v:j for j,v in enumerate(set(codes)) }
    for row in rows:
        for cn, v in enumerate(row):
            row[cn] = float(v) if cn<18 else codes_h[v] if cn==19 else 0
    arows = utils.shuffle(np.array(rows), random_state=0)
    x_row_indexes = [0,1,2,3,4,5,6, 8,9,10,11,12,13,14,15,16,17,19]
    x_head = np.array(head)[x_row_indexes]
    X = arows[:, x_row_indexes]
    Y = arows[:, [7]]
    rng, l = np.arange(len(X)), len(X)/3
    train, test = rng > l, rng <= l
    return X, Y, x_head, train, test

def mode_check_rows():
    rows, head = load_rows()
    print x_head
    phones = [row[20] for row in rows]
    print len(phones), len(set(phones)) #checking phones are uniq
    codes = [row[19] for row in rows]
    states = [row[18] for row in rows]
    print set(codes), set(states) #codes '415', '510', '408' all from California

    X, Y, x_head, train, test = rows_to_predictor_response(rows, head)
    print head, x_head, head[7]
    print X.shape, Y.shape
    print len(Y[train]), len(Y[test])
    print np.count_nonzero(Y[train]), np.count_nonzero(Y[test])
    print np.count_nonzero(Y)/float(len(Y))
    print np.count_nonzero(Y[train])/float(len(Y[train]))
    print np.count_nonzero(Y[test])/float(len(Y[test]))

def plot_rocs(nm,color,y_true,y_pred):
    fpr, tpr, thresholds = metrics.roc_curve(y_true=y_true, y_score
        =y_pred)
    print metrics.auc(fpr, tpr)
    if len(fpr)<4: print fpr, tpr
    path = "out/roc."+nm+".npz"
    np.savez(path,fpr=fpr, tpr=tpr, thresholds=thresholds, color=np.array([color]))
    plt.title('ROC␣curves')
    plt.xlabel('FPR')
    plt.ylabel('TPR')
    for fn in glob.glob("out/roc.*.npz"):
        v = np.load(fn)
        plt.plot(v['fpr'], v['tpr'], v['color'][0])
    plt.savefig("out/roc.png")
    plt.cla()


def mode_pca():
    rows, head = load_rows()
    X, Y, x_head, train, test = rows_to_predictor_response(rows, head)

    pca = decomposition.PCA(n_components=13)
    re_pipeline = pipeline.Pipeline([ ('scaler',preprocessing.Scaler()), ('pca',pca) ])
    pc = re_pipeline.fit_transform(X[train])

    churn = Y[train,0] > 0.5
    for i in range(1,13):
        plt.title('PCA␣scores')
        plt.xlabel('pc[0]')
        plt.ylabel('pc['+str(i)+']')
        plt.plot(pc[:,0], pc[:,i], 'go')
        plt.plot(pc[churn,0], pc[churn,i], 'ro')
        plt.savefig("out/churn_scores_0_"+str(i)+".png")
        plt.cla()

    loadings = pca.components_
    for i in range(1,13):
        plt.title('PCA␣loadings')
        plt.xlabel('pc[0]')
        plt.ylabel('pc['+str(i)+']')
        plt.plot(loadings[0], loadings[i], 'go')
        for j,l in enumerate(loadings[[0,i]].T):
            plt.annotate(x_head[j],l); #-(j%3)*0.02
            #print x_head[i],l
        plt.savefig("out/churn_loadings_0_"+str(i)+".png")
        plt.cla()

    y_scaler = preprocessing.Scaler(with_std=False)
    linre = linear_model.LinearRegression()
    linre.fit(X=pc, y=y_scaler.fit_transform(Y[train,0]))

    y_pred = y_scaler.inverse_transform( linre.predict( re_pipeline.transform(X[test]) ) )

    plot_rocs('pca','b-',Y[test,0],y_pred)

```

```
def mode_pls():
    rows, head = load_rows()
    X, Y, x_head, train, test = rows_to_predictor_response(rows, head)

    re = pls.PLSRegression(n_components=11)
    re.fit(X=X[train], Y=Y[train])
    y_pred = re.predict( X[test] )[:,0]

    plot_rocs('pls','b-',Y[test,0],y_pred)

def mode_linre():
    rows, head = load_rows()
    X, Y, x_head, train, test = rows_to_predictor_response(rows, head)

    linre = linear_model.LinearRegression()
    linre.fit(X=X[train], y=Y[train,0])
    y_pred = linre.predict( X[test] )

    plot_rocs('linre','b-',Y[test,0],y_pred)

def mode_tree():
    rows, head = load_rows()
    X, Y, x_head, train, test = rows_to_predictor_response(rows, head)

    clf = tree.DecisionTreeClassifier(max_depth=5)#, min_samples_leaf?
    clf.fit(X=X[train], y=Y[train,0]) #*2-1
    #y_pred = clf.predict( X[test] )
    y_pred = clf.predict_proba( X[test] )[:,1]

    plot_rocs('tree','g-',Y[test,0],y_pred)

    tree.export_graphviz(clf, out_file='dtree.graphviz', feature_names=x_head)
    #convert using: dot dtree.graphviz -Tpng > dtree2.png

mode_linre()
mode_pca()
mode_pls()
mode_tree()
```