Question1

Issue with data:

Categorical data

- We have two features that have missing values. For one feature, we have 74% of missing values so we omitted. For another feature, we have filled missing values with frequent ones with its churn type.

- We have features that have same class with different names. Like 's' and 'suburban'; 't' and 'true'. We have replaced the correlated classes with same meaningful one. Like replaced 's' with 'suburban'; 't' with 'true'.

Numerical data

- We have features with negative values. Model can be sensitive towards negative values. We plotted distribution of the feature to figure out a strategy to deal with negative values. For one feature, we just replaced negative values with zero, as the number of negative values is less. For other features, we used min(negative value) while logarithmic transformation.

- By looking at the statistics of the numerical data, we derived that we have skewed values. Usually for features, having outliers, difference between mean and standard deviation would be considerable. We used Logarithmic transformation to correct the skewness.

- We have different features with different scales. Having different scales would harm the model, so we used minmax scaler which makes all the features value between 0 and 1.

Normalization

- For most of the features, the counts of different classes are similar when churn=true and churn=false. This might establish similar patterns for similar sets of data with different target class.

- We can use normalized features as a strategy to break above patterns. We used count and mean of categorical values after one-hot encoding. We then normalized the counts. Normalization was useful.

Question2

Classifiers: We used different classifiers to see the performance.

- Decision tree: This is a linear classifier. We have high number of features around 40. Decision tree might not perform better with more number of features. We have Training accuracy = 0.75471 and Testing accuracy = 0.73700

- Random Forest: This is like a multitude of decision trees running with bootstrapping the features and samples. This also a linear classifier and might not perform well in the presence of high dimensions. We have Training accuracy = 0.73786 and Testing accuracy = 0.74200

- K neighbors: This is a non-linear classifier. We have Training accuracy = 0.77157 and Testing accuracy = 0.70900. We can see little less testing accuracy than the training. We can see little overfitting on Training data.

- SVM: This is a linear classifier and good for high dimensional data. We have Training accuracy = 0.73757 and testing accuracy = 0.74000

- Neural net classifier: This could be linear or non-linear basing on the activation function. We use Relu which is piece-wise linear activation function. This could handle high dimensional data. We have Training accuracy = 0.74271 and Testing accuracy = 0.74033

Question3

From above accuracy information, I think SVM and Neural Network classifier are good to evaluate the given data for following reasons,

- I think, our data has underlying linearity between features. Both SVM and Neural network are linear classifiers. At high dimensions also, SVM behaves as linear classifier. Both the classifiers can handle high dimensional data. We have consistent training and testing accuracies.

- Out of SVM and neural net classifiers, I would prefer neural net classifier as it has consistent training and testing accuracies.

Extra credit

According to new information

- Service to churner costs then company 700dol
- Excluding a non-churner costs 100dol

From above information, we can say that

- Classifying a churner as non-churner would cost 700dol
- Classifying a non-churner as a churner would cost 100dol

In order to include this information into our model, we are going to use weights to the classes in our target variable 'Churn'. Ratio would 1:7, we would make our model penalize 7 times more when a churner is classified as non-churner than the vice-versa.

```
#change the classes of target variable
X_train = X_train_h
X_test = X_test_h
y_train.loc[y_train == False] = 0
y_train.loc[y_train == True] = 1
y_test.loc[y_test == False] = 0
y_test.loc[y_test == True] = 1
cw = {1:7,0:1}
clf_s = SVC(kernel='poly',degree=2,gamma='auto',random_state=4,class_weight=cw
)
clf_fit_train_s=clf_s.fit(X_train,y_train)
clf_predict_train_s = clf_fit_train_s.predict(X_train)
clf_predict_test_s = clf_fit_train_s.predict(X_test)
resultsv = {}
resultsv['acc_train'] = accuracy_score(y_train, clf_predict_train_s)
resultsv['acc_test'] = accuracy_score(y_test, clf_predict_test_s)
resultsv['f_train'] = fbeta_score(y_train, clf_predict_train_s, average='mic
ro', beta=1)
resultsv['f_test'] = fbeta_score(y_test, clf_predict_test_s, average='micr
o', beta=1.5)
print "Accuracy for SVM Classifier-Training, Test sets: %.5f, %.5f" %(resultsv
['acc_train'], resultsv['acc_test'])
print "----------------------------------------------------------------------
"
```

```
Accuracy for SVM Classifier-Training, Test sets: 0.50014, 0.49967
------------------------------------------------------------------------
```

Analysis:

- We can see introducing class weights had decreased the accuracy of the model.
- To increase, I think we need to work on strategy to include new features which could make the model understand the domain knowledge of the data.