Reports on Spark lab 3, performed with Apache Spark 1.6.2, with nice picture at the end.

Disclaimer: I'm totally not happy with converting data back and forth from RDD to DataFrame either, but API leaves a lot to be desired. Since it's hard to follow its transformations, I'll refer to it as "dataset" without any specification.

1. Preparing data

As manual analysis shows (performed with pandas, by the way — Scala is too clumsy for the ad-hoc analysis), dataset contains a significant number of NaN values. In fact, almost every entry contains NaN in the last column, but in this case we can safely (-ish) assume, that NaN here is just a zero. As for other columns, we cannot make any sensible assumptions about their true value and should drop them.
Since dataset contains categorical variables, we should make it more suitable for predicting algorithms. First of all every categorical variable should be transformed via one-hot encoding into sequence of length equal to number of levels with exactly one 1 and length-1 zeroes. This action involves some pretty heavy transformation of dataset to its OHE map, but it's done only once and then aforementioned map is broadcasted across cluster.
Note: yes, MlLib has OneHotEncoder, but I found it not very convenient to use.

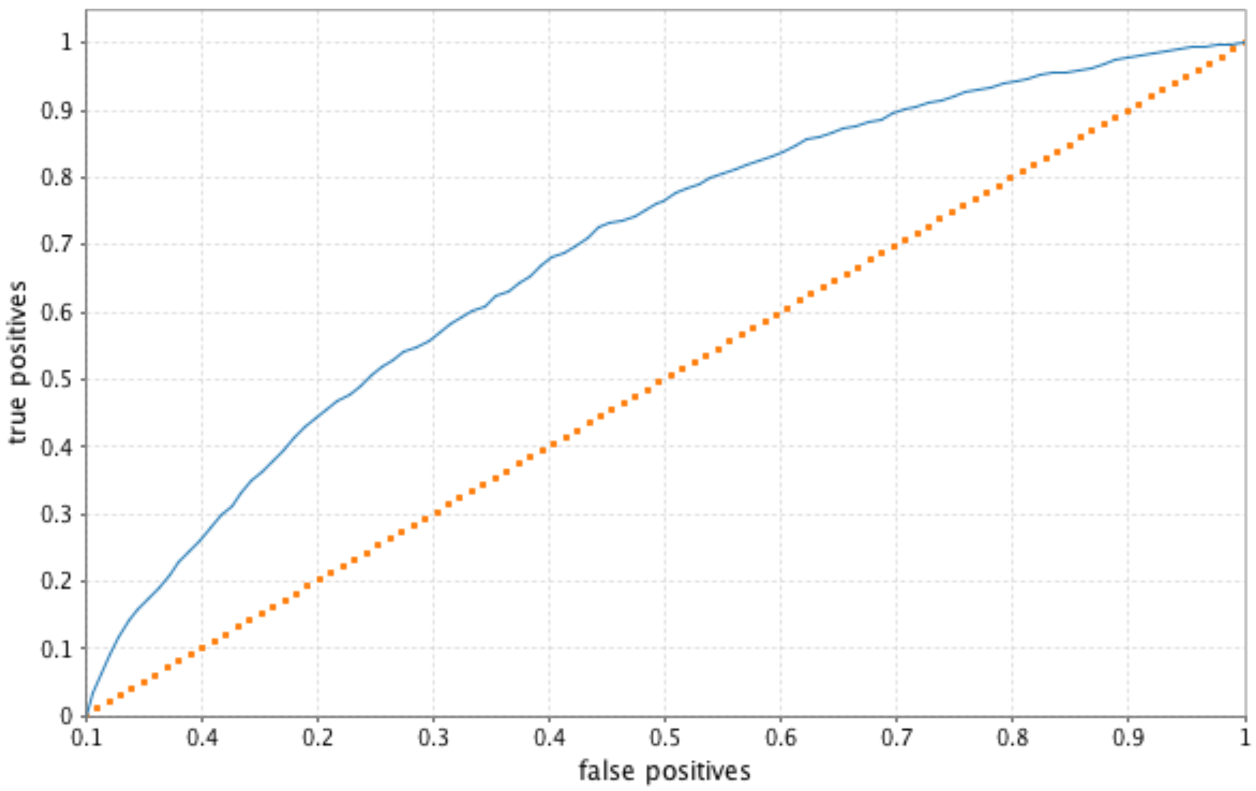
At this moment we have a dataset with very long rows and very many zeroes in them. That's not very efficient way to store data, so it would be reasonable to convert rows into sparse form to store only meaningful (i.e., non-zero) elements. Luckily, LabeledPoint class, that is used with learning algorithms is happy to store sparse vectors.
if we dealt with larger dataset, it would be beneficial to reduce its dimensionality using ChiSqSelector to leave only the most significant features, but since we have <500 features to analyse (and ChiSqSelector requires special handling of any continuous feature) we're good to go.

PS PropertyDesciptionEN.txt sucks on an epic scale. Most of the values marked as numeric are, in fact, categorical. I mean, there's no way for property like "Registered address and postal address match" (that has **two** levels) to be numeric.

2. Choosing an algorithm
As we're dealing with simple binomial classification, virtually any predicting algorithm will do.
And, thanks to API inconsistency there's no obvious way to extract AUC from anything except LogisticRegression, so logistic regression with some grid search and cross-validation is our algorithm of choice for today.
Grid parameters cover multiple kinds of logistic regressions, from ridge to Lasso, CrossValidator tuned to perform ten-fold validation (10% test set seems reasonable).

3. Learning
Since our chosen metric is AUC, we don't need explicit test dataset, CrossValidator will take care of that, so we'll train our model on entire dataset. Not much to write here 'cause after all is set and done there's quite a lot of waiting.

4. Conclusions
According to AUC (0.69), our model performs slightly more effective than random draw, so it could be used in absence of any better implementation.



true positives
false positives
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0.1
0.4
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

Pic.1. ROC