Table of Contents

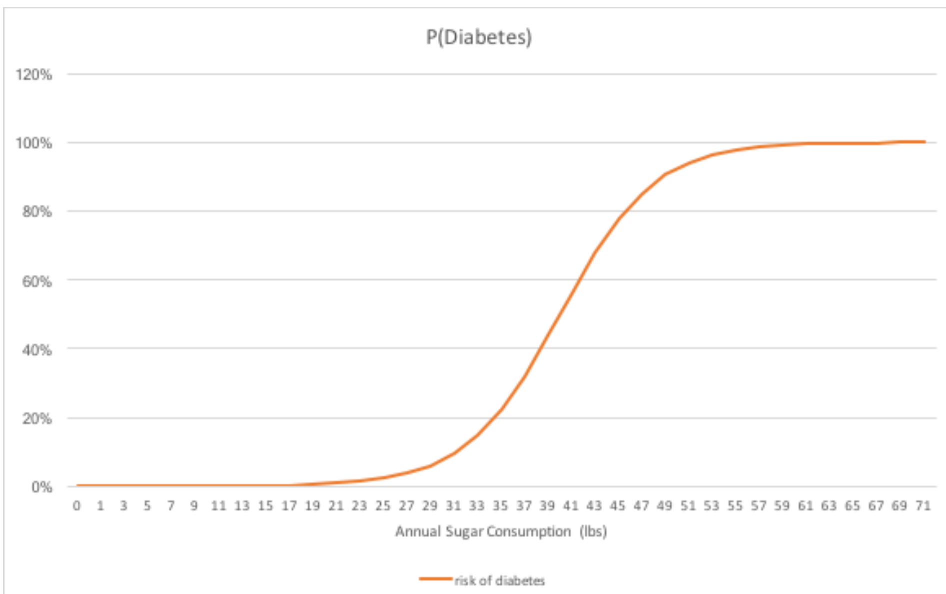
Motivation

Our motivation in this exercise is slightly different from the last two. Whereas previously the goal was to target a high f1 or accuracy score, here we're focusing on interpretation. As a result we'll spend more time focusing on interpreting model output instead of making a better model.

Data Preparation

Transformations

As before, we're going to convert data types and impute null values to means. However, **we won't log- or exp-transform our features** because doing so makes interpretation confusing. Take this fake example where we relate sugar consumption to the risk of diabetes:



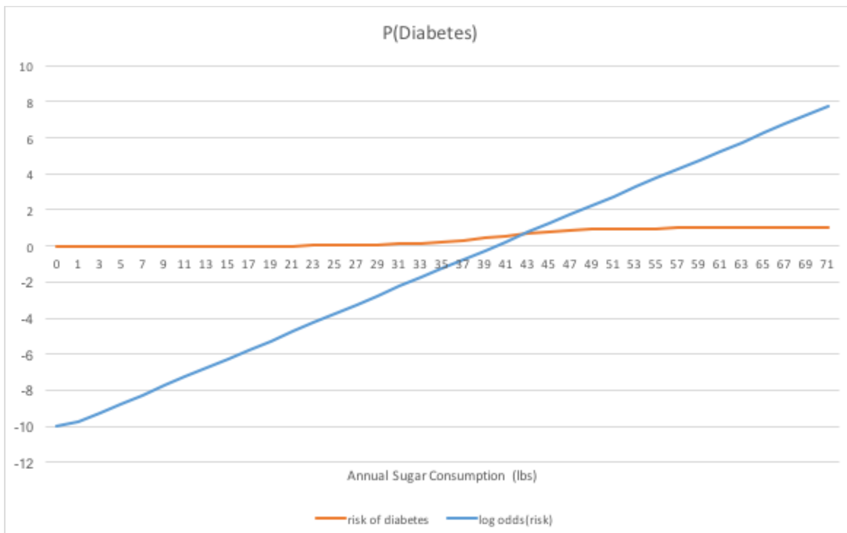
We can make a statement like, "a an increase of sugar consumption from 31 to 33 lbs leads to a 10% increase in the risk of diabetes." But the statement changes at various levels of x, because the curve is not linear. The logistic or sigmoid function on which this curve is based is this:

$$F(x) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}}$$

If you transform the above equation, you get a linear equation set against the “log odds” of an outcome:

$$\ln\left(\frac{F(x)}{1-F(x)}\right) = \beta_0 + \beta_1 x,$$

Interpreting the log odds, you can make an alternative statement like, “a one unit increase in annual sugar consumption leads to one unit increase in the log odds of having diabetes,” because the relationship between x and log odds (blue line) is linear. A one unit increase in x always has the same impact on the log odds of an outcome.



We will go into log-odds in more detail in the model interpretation section of this guide.

Dummy Variables

It's also worth noting that since we're focused on interpreting the model, we're going to do one-hot/dummy encoding but remove one of the categories. The reason for this include interpretation, multicollinearity, and the dummy variable trap. Think about the case where there's just two categories, male and female. We represent both by selecting just one of the fields, because using both is redundant (when a person isn't male, they're necessarily female):

male	female
1	0
1	0
1	0
0	1
0	1

Now let's say we introduced a new category, transgender, the same logic applies. When a person isn't male or transgender, they're necessarily female. Having three variables representing three categories leads to the above problems.

male	female	trans
1	0	0
1	0	0
0	0	1
0	1	0
0	1	0

This video provides further explanation of why you drop one of the categories when doing dummy encoding: https://www.youtube.com/watch?v=9yTui_LoSOc.

Feature Selection

ANOVA for Univariate Feature Selection

One measure of the linear strength of a relationship is the F-Statistic from ANOVA. It tells you the strength of the relationship between both categorical and continuous variables against your target class. If you suspect a non-linear relationship, you can use mutual_info_classif instead.

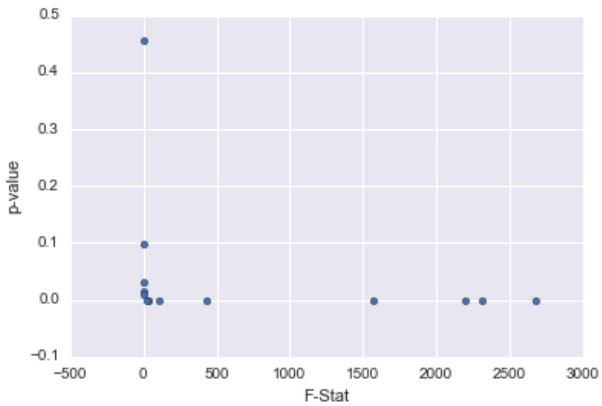
```
from sklearn.feature_selection import f_classif, mutual_info_classif
X = Df.drop('churn', axis=1)
y = Df['churn']

# get anova results
fc = f_classif(X, y)
DfAnova = pd.DataFrame(np.vstack(fc).T, index=X.columns, columns=['F-Stat', 'p-value'])
print np.round(DfAnova.sort_values('F-Stat', ascending=False), 4)
```

```
                          F-Stat  p-value
phone_Android          2677.7169   0.0000
trips_in_first_30_days 2317.2939   0.0000
black_car_user         2193.3861   0.0000
city_Astapor           1573.1445   0.0000
avg_dist                434.1249   0.0000
city_Winterfell         108.8594   0.0000
avg_rating_by_driver     35.7971   0.0000
days_from_first_signup   20.4310   0.0000
surge_pct                 6.9589   0.0083
avg_rating_of_driver      5.8635   0.0155
weekday_pct               4.6979   0.0302
phone_Other               2.7521   0.0971
avg_surge                 0.5555   0.4561
```

The results show an F-Statistic which is a test statistic on the F-Distribution, and p-value, which tells you how significant a feature is or isn't. High F-Statistics are good and low p-values are good (usually below 0.005).

```
DfAnova = pd.DataFrame(data=np.vstack(fc).T, columns=['F-Stat', 'p-value'])
DfAnova.plot(kind='scatter', x='F-Stat', y='p-value')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x12dd7bf50>
```



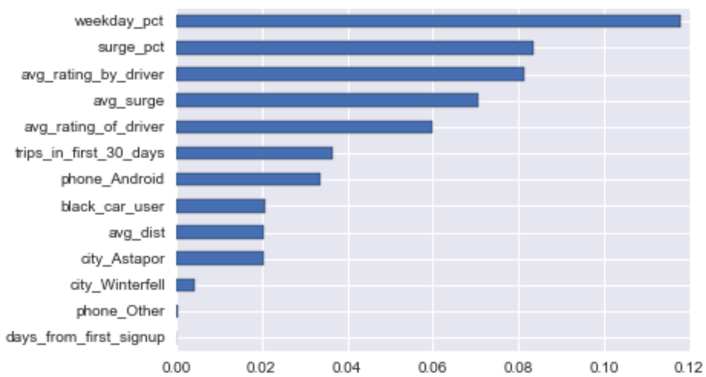
Here we see phone_Android, trips, black_car, city_Astapor, avg_dist are strongly related to churn (very high F-Stat). Avg_surge, phone_other, weekday_pct, and avg_rating_of_driver are not strongly related (high p-value and low F-Stat).

Mutual Information

Another feature selection technique that captures non-linear relationships (good for Random Forest and Neural Nets) is Mutual Information. You can read more at the link above, but it's worth noting that Mutual Information and ANOVA point to very different features.

```
# get mutual_info results
mc = mutual_info_classif(X, y)
pd.Series(data=mc, index=X.columns).sort_values().plot(kind='barh')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x13cb064d0>
```



We will go into more detail on how to capture these non-linear relationships in Logistic Regression at the end.

Multi-Collinearity

In model building there's an issue called multi-collinearity, in which two highly correlated variables "steal signal" from each other. Because the variables are so related to each other, the model doesn't know whether to attribute predictive value to one or the other. If a customer never has surge pricing, then avg_surge = 1 and surge_pct = 0. You can see the correlation coefficient is very high between these two at 79%:

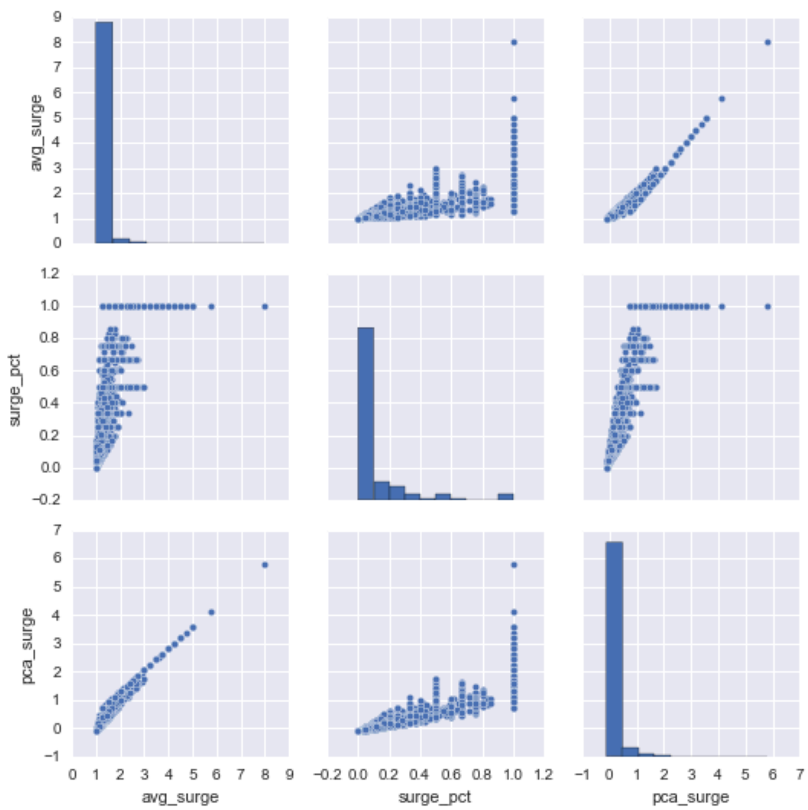
```python
# look at correlations within features
(np.round(Df.corr(), 4)*100).astype(np.int)
```

	avg_dist	avg_rating_by_driver	avg_rating_of_driver	avg_surge	black_car_user	surge_pct
avg_dist	100	7	2	-8	3	-10
avg_rating_by_driver	7	100	10	1	0	2
avg_rating_of_driver	2	10	100	-2	0	0
avg_surge	-8	1	-2	100	-7	79
black_car_user	3	0	0	-7	100	-10
surge_pct	-10	2	0	79	-10	100
trips_in_first_30_days	-13	-3	-1	0	11	0
weekday_pct	10	2	1	-11	3	-14
days_from_first_signup	1	0	0	0	0	0
churn	9	2	1	0	-20	-1
city_Astapor	-4	3	2	3	2	4
city_Winterfell	3	-9	-9	-2	-5	-6
phone_Android	2	0	2	0	-7	-1
phone_Other	2	0	0	-1	4	-1

One way to address multicollinearity is to create a feature that that combines the two original ones, as with principle components analysis which reduces the two features to just one shared one. When you do this, you find the PCA feature is highly correlated with both surge features, so serves as a potentially good go-between:

```
from sklearn.decomposition import PCA

DfMc = pd.DataFrame()

# create new df for Mc columns
DfMc[['avg_surge', 'surge_pct']] = Df[['avg_surge', 'surge_pct']]

# get priciple component for avg_surge and surge_pct
pca = PCA(n_components=1)
DfMc['pca_surge'] = pca.fit_transform(DfMc.loc[:, ['avg_surge', 'surge_pct']])

# analyze ther variables relative to each other
print np.round(DfMc.corr(), 4)*100
sns.pairplot(DfMc)
```

```
           avg_surge  surge_pct  pca_surge
avg_surge     100.00      79.36      95.94
surge_pct      79.36     100.00      93.30
pca_surge      95.94      93.30     100.00

<seaborn.axisgrid.PairGrid at 0x139fa2050>
```



But since avg_surge is not strongly related to churn and the principle component of avg_surge and surge_pct isn't either, we will drop it instead of creating a feature that captures both surge variables:

```
fc = f_classif(DfMc, y)

DfAnova = pd.DataFrame(np.vstack(fc).T, index=DfMc.columns, columns=['F-Stat', 'p-value'])
DfAnova = np.round(DfAnova, 4)
DfAnova
```

	F-Stat	p-value
avg_surge	0.5555	0.4561
surge_pct	6.9589	0.0083
pca_surge	0.6114	0.4343

```
# keep surge_pct, don't use the PCA term
Df = Df.drop(['avg_surge'], axis=1)
```

Model Fitting

Backward Stepwise Regression

Backward Stepwise Regression starts with the **full model**, where all features included, then takes features away one-by-one until all of the features fit a certain criteria. **Instead of using scikit-learn, we will use statsmodels** because it has many more diagnostics for model interpretation.

```
import statsmodels.api as sm
logit = sm.Logit(y, X)
result = logit.fit()
print result.summary()
```

```
Optimization terminated successfully.
         Current function value: 0.557103
         Iterations 6
                           Logit Regression Results
==============================================================================
Dep. Variable:                      y   No. Observations:                50000
Model:                          Logit   Df Residuals:                    49987
Method:                           MLE   Df Model:                           12
Date:                Mon, 19 Dec 2016   Pseudo R-squ.:                  0.1586
Time:                        16:57:24   Log-Likelihood:                -27855.
converged:                       True   LL-Null:                       -33106.
                                        LLR p-value:                     0.000
==========================================================================================
                             coef    std err          z      P>|z|      [95.0% Conf. Int.]
------------------------------------------------------------------------------------------
avg_dist                   0.0363      0.002     17.485      0.000         0.032     0.040
avg_rating_by_driver       0.1587      0.023      6.943      0.000         0.114     0.204
avg_rating_of_driver       0.0576      0.018      3.166      0.002         0.022     0.093
black_car_user            -0.8682      0.021    -40.668      0.000        -0.910    -0.826
surge_pct                 -0.2305      0.053     -4.313      0.000        -0.335    -0.126
trips_in_first_30_days    -0.1204      0.003    -34.745      0.000        -0.127    -0.114
weekday_pct                0.0154      0.029      0.537      0.591        -0.041     0.072
days_from_first_signup     0.0061      0.001      5.145      0.000         0.004     0.008
city_Astapor               1.7226      0.030     58.182      0.000         1.665     1.781
city_Winterfell            1.2151      0.027     44.864      0.000         1.162     1.268
phone_Android              1.0863      0.025     44.149      0.000         1.038     1.135
phone_Other                0.6267      0.116      5.410      0.000         0.400     0.854
intercept                 -1.5848      0.138    -11.483      0.000        -1.855    -1.314
==========================================================================================
```

LLR p-value tells you whether or not the model is statistically significant and is derived from Log-Likelihood and LL-Null. Since the value is very close to 0, we know this model is predictive.

The **Z-scores** and **p-values** for each of the coefficients tells you how significant each feature is. High Z's and low p's (close to 0) are very good. Immediately we notice weekday_pct has a very high p-value, so we drop it:

```
# drop weekday_pct
logit = sm.Logit(y, X.drop(['weekday_pct'], axis=1))
result = logit.fit()
print result.summary()
```

```
Optimization terminated successfully.
         Current function value: 0.557105
         Iterations 6
                           Logit Regression Results
==============================================================================
Dep. Variable:                      y   No. Observations:                50000
Model:                          Logit   Df Residuals:                    49988
Method:                           MLE   Df Model:                           11
Date:                Mon, 19 Dec 2016   Pseudo R-squ.:                  0.1586
Time:                        16:28:45   Log-Likelihood:                -27855.
converged:                       True   LL-Null:                       -33106.
                                        LLR p-value:                     0.000
==========================================================================================
                             coef    std err          z      P>|z|      [95.0% Conf. Int.]
------------------------------------------------------------------------------------------
avg_dist                   0.0365      0.002     17.630      0.000         0.032     0.041
avg_rating_by_driver       0.1589      0.023      6.953      0.000         0.114     0.204
avg_rating_of_driver       0.0576      0.018      3.169      0.002         0.022     0.093
black_car_user            -0.8681      0.021    -40.666      0.000        -0.910    -0.826
surge_pct                 -0.2344      0.053     -4.425      0.000        -0.338    -0.131
trips_in_first_30_days    -0.1202      0.003    -34.781      0.000        -0.127    -0.113
days_from_first_signup     0.0061      0.001      5.148      0.000         0.004     0.008
city_Astapor               1.7223      0.030     58.185      0.000         1.664     1.780
city_Winterfell            1.2146      0.027     44.874      0.000         1.162     1.268
phone_Android              1.0862      0.025     44.147      0.000         1.038     1.134
phone_Other                0.6273      0.116      5.415      0.000         0.400     0.854
intercept                 -1.5768      0.137    -11.492      0.000        -1.846    -1.308
==========================================================================================
```

Notice that LLR p-value is unchanged and Log-Likelihood is exactly the same, so dropping the feature didn't have a material impact on our model. Let's try this one more time with avg_rating_of_driver:

```
# drop weekday_pct and avg_rating_of_driver
logit = sm.Logit(y, X.drop(['weekday_pct', 'avg_rating_of_driver'], axis=1))
result = logit.fit()
print result.summary()
```

```
Optimization terminated successfully.
         Current function value: 0.557205
         Iterations 6
                           Logit Regression Results
==============================================================================
Dep. Variable:                      y   No. Observations:                50000
Model:                          Logit   Df Residuals:                    49989
Method:                           MLE   Df Model:                           10
Date:                Mon, 19 Dec 2016   Pseudo R-squ.:                  0.1584
Time:                        16:28:45   Log-Likelihood:                -27860.
converged:                       True   LL-Null:                       -33106.
                                        LLR p-value:                     0.000
==========================================================================================
                             coef    std err          z      P>|z|      [95.0% Conf. Int.]
------------------------------------------------------------------------------------------
avg_dist                   0.0366      0.002     17.694      0.000         0.033     0.041
avg_rating_by_driver       0.1654      0.023      7.273      0.000         0.121     0.210
black_car_user            -0.8683      0.021    -40.681      0.000        -0.910    -0.827
surge_pct                 -0.2356      0.053     -4.448      0.000        -0.339    -0.132
trips_in_first_30_days    -0.1203      0.003    -34.790      0.000        -0.127    -0.114
days_from_first_signup     0.0061      0.001      5.153      0.000         0.004     0.008
city_Astapor               1.7184      0.030     58.116      0.000         1.660     1.776
city_Winterfell            1.2064      0.027     44.793      0.000         1.154     1.259
phone_Android              1.0880      0.025     44.229      0.000         1.040     1.136
phone_Other                0.6287      0.116      5.428      0.000         0.402     0.856
intercept                 -1.3389      0.115    -11.669      0.000        -1.564    -1.114
==========================================================================================
```

Notice that Log-Likelihood is mostly unchanged (only 5 higher). Also notice the most significant features, the ones with the highest Z-scores, are **city, phone, black_car_user, and trips_in_first_30_days.**

Regularization

Another approach we can take to feature selection is regularization. Instead of manually adding and removing features, we can penalize the cost function to restrict the weight it assigns to features. We will go into much more detail with this in Appendix 2, but for now you can see that the regularized model still favors the features we originally identified as important using **ANOVA**:

```
logit = sm.Logit(y, X, axis=1)
result = logit.fit_regularized(alpha=1/0.01, method='l1')
print result.summary()
```

```
Optimization terminated successfully.    (Exit mode 0)
            Current function value: 0.568510028361
            Iterations: 47
            Function evaluations: 51
            Gradient evaluations: 47
                           Logit Regression Results
==============================================================================
Dep. Variable:                      y   No. Observations:                50000
Model:                          Logit   Df Residuals:                    49991
Method:                           MLE   Df Model:                            8
Date:                Mon, 19 Dec 2016   Pseudo R-squ.:                  0.1552
Time:                        16:29:26   Log-Likelihood:                -27968.
converged:                       True   LL-Null:                       -33106.
                                        LLR p-value:                     0.000
==========================================================================================
                             coef    std err          z      P>|z|      [95.0% Conf. Int.]
------------------------------------------------------------------------------------------
avg_dist                   0.0351      0.002     17.351      0.000         0.031     0.039
avg_rating_by_driver      -0.0303      0.015     -1.958      0.050        -0.061  2.73e-05
avg_rating_of_driver      -0.0464      0.016     -2.988      0.003        -0.077    -0.016
black_car_user            -0.8102      0.021    -38.873      0.000        -0.851    -0.769
surge_pct                       0        nan        nan        nan           nan       nan
trips_in_first_30_days    -0.1214      0.003    -35.479      0.000        -0.128    -0.115
weekday_pct                     0        nan        nan        nan           nan       nan
days_from_first_signup     0.0038      0.001      3.270      0.001         0.002     0.006
city_Astapor               1.5102      0.029     52.927      0.000         1.454     1.566
city_Winterfell            1.0118      0.026     39.585      0.000         0.962     1.062
phone_Android              1.0040      0.024     41.661      0.000         0.957     1.051
phone_Other                     0        nan        nan        nan           nan       nan
intercept                       0        nan        nan        nan           nan       nan
==========================================================================================
```

Model Interpretation

Odds Ratio

The most important tool for interpreting Logistic Regression model output is the **odds ratio**. The Logistic Function isn't linear, but the log-odds of it is. You should read about the Odds Ratio in detail here.

```
# selected model
logit = sm.Logit(y, X.drop(['weekday_pct', 'avg_rating_of_driver'], axis=1))
result = logit.fit()
print result.summary()
```

```
Optimization terminated successfully.
         Current function value: 0.557143
         Iterations 6
                           Logit Regression Results
==============================================================================
Dep. Variable:                      y   No. Observations:                50000
Model:                          Logit   Df Residuals:                    49988
Method:                           MLE   Df Model:                           11
Date:                Mon, 19 Dec 2016   Pseudo R-squ.:                  0.1585
Time:                        19:34:49   Log-Likelihood:                -27857.
converged:                       True   LL-Null:                       -33106.
                                        LLR p-value:                     0.000
==========================================================================================
                             coef    std err          z      P>|z|      [95.0% Conf. Int.]
------------------------------------------------------------------------------------------
avg_dist                   0.0366      0.002     17.686      0.000         0.033     0.041
avg_rating_by_driver       0.1656      0.023      7.278      0.000         0.121     0.210
avg_surge                  0.1948      0.080      2.443      0.015         0.039     0.351
black_car_user            -0.8692      0.021    -40.714      0.000        -0.911    -0.827
surge_pct                 -0.4070      0.087     -4.670      0.000        -0.578    -0.236
trips_in_first_30_days    -0.1202      0.003    -34.764      0.000        -0.127    -0.113
days_from_first_signup     0.0061      0.001      5.182      0.000         0.004     0.008
city_Astapor               1.7159      0.030     58.000      0.000         1.658     1.774
city_Winterfell            1.2027      0.027     44.592      0.000         1.150     1.256
phone_Android              1.0883      0.025     44.237      0.000         1.040     1.137
phone_Other                0.6292      0.116      5.432      0.000         0.402     0.856
intercept                 -1.5315      0.139    -10.993      0.000        -1.805    -1.258
==========================================================================================
```

phone_Android has coefficient = 1.0883. The way to read this is, all other factors held equal, Android users are exp(1.0883) more likely to churn than iPhone users (the base class in this example). That is, their odds of churning are 2.969 versus iPhone users.

trips_in_first_30_days has a slightly different interpretation since it's a continuous variable. The way to read this is, all other factors held equal, a one unit increase in trips leads to an exp(-0.1202) increase in the likelihood of churn. This translates to 0.8867 increase or -0.1133 decrease in the odds of churning.

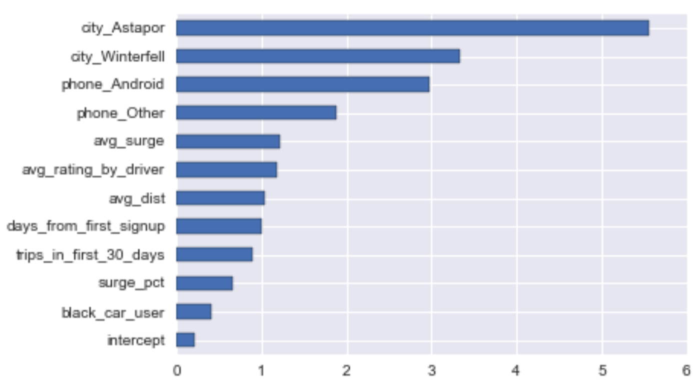
You can get a sense of which features most affect the odds of churn by looking at the full set of odds ratios. Keep in mind binary features can only affect odds by 1x, whereas continuous ones such as trips_in_first_30_days have a multiplicative effect. For better understanding of what this means, compare difference in p(churn) between different levels of trips and phone_Android in the Prediction Simulation section.

```
np.exp(result.params)
```

```
avg_dist                  1.037239
avg_rating_by_driver      1.180086
avg_surge                 1.215079
black_car_user            0.419274
surge_pct                 0.665616
trips_in_first_30_days    0.886703
days_from_first_signup    1.006114
city_Astapor              5.561830
city_Winterfell           3.329234
phone_Android             2.969192
phone_Other               1.876129
intercept                 0.216202
dtype: float64
```

```
# odds ratio
np.round(np.exp(result.params), 2).sort_values(ascending=True).plot(kind='barh')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x134e19850>
```



You can also get 95% confidence intervals of the odds ratio estimates:

```python
# odds ratio confidence intervals
params = result.params
conf = result.conf_int()
conf['OR'] = params
conf.columns = ['2.5%', '97.5%', 'OR']
print np.round(np.exp(conf), 2).sort_values('OR', ascending=False)
```

```
                         2.5%  97.5%    OR
city_Astapor             5.25   5.89  5.56
city_Winterfell          3.16   3.51  3.33
phone_Android            2.83   3.12  2.97
phone_Other              1.50   2.35  1.88
avg_surge                1.04   1.42  1.22
avg_rating_by_driver     1.13   1.23  1.18
avg_dist                 1.03   1.04  1.04
days_from_first_signup   1.00   1.01  1.01
trips_in_first_30_days   0.88   0.89  0.89
surge_pct                0.56   0.79  0.67
black_car_user           0.40   0.44  0.42
intercept                0.16   0.28  0.22
```

Prediction Simulation

Another less confusing but also less descriptive tool for understanding the factors that influence churn is by simulation. Using the **itertools** package, we can create the simulated data using a fea key features (like black_car_user, city, phone, and trips_in_first_30_days), and plot them.

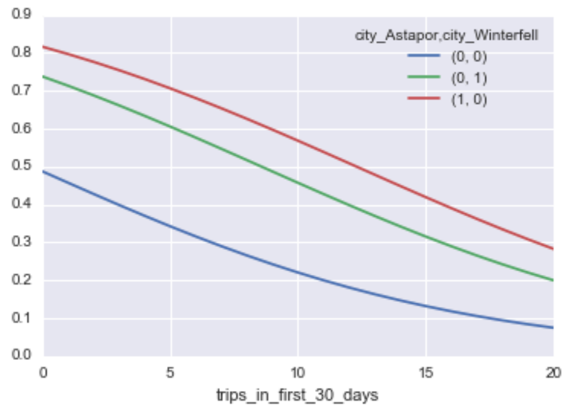
```python
# isolate features of interest from original df
sim_phone = DfTypes['phone'].unique()
sim_city = DfTypes['city'].unique()
sim_blackcar = DfTypes['black_car_user'].unique()
sim_trips = range(21) # based on sns.kdeplot(DfTypes['trips_in_first_30_days'])
```

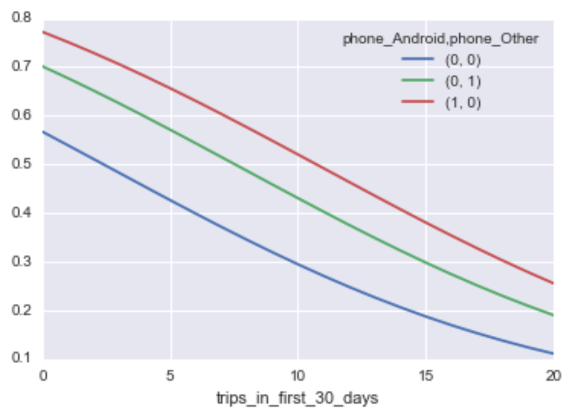
```python
from itertools import combinations

# create simulated df
cartesian = itertools.product(*[sim_phone, sim_city, sim_blackcar, sim_trips])
cols = ['phone', 'city', 'black_car_user', 'trips_in_first_30_days']
DfSim = pd.DataFrame.from_records(list(cartesian), columns=cols)

# convert categories to dummies
DfSim = pd.get_dummies(DfSim)
DfSim = DfSim.drop(["city_King's Landing", 'phone_iPhone'], axis=1)

# fit model
X_fit = Df[DfSim.columns.tolist()]
X_fit['intercept'] = 1
logit = sm.Logit(y, X_fit)
result = logit.fit()
print result.summary()

# use model for predictions
X_sim = DfSim
X_sim['intercept'] = 1
y_pred_sim = result.predict(X_sim)
```

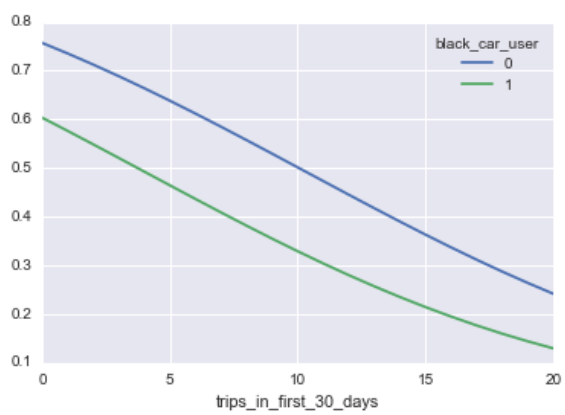
```
Optimization terminated successfully.
         Current function value: 0.562027
         Iterations 6
                           Logit Regression Results
==============================================================================
Dep. Variable:                      y   No. Observations:                50000
Model:                          Logit   Df Residuals:                    49993
Method:                           MLE   Df Model:                            6
Date:                Mon, 19 Dec 2016   Pseudo R-squ.:                  0.1512
Time:                        19:51:47   Log-Likelihood:                -28101.
converged:                       True   LL-Null:                       -33106.
                                        LLR p-value:                     0.000
==========================================================================================
                             coef    std err          z      P>|z|      [95.0% Conf. Int.]
------------------------------------------------------------------------------------------
black_car_user            -0.8253      0.021    -39.342      0.000        -0.866    -0.784
trips_in_first_30_days    -0.1306      0.003    -37.847      0.000        -0.137    -0.124
phone_Android              1.0889      0.024     44.473      0.000         1.041     1.137
phone_Other                0.6728      0.115      5.855      0.000         0.448     0.898
city_Astapor               1.6616      0.029     56.924      0.000         1.604     1.719
city_Winterfell            1.1752      0.027     44.334      0.000         1.123     1.227
intercept                 -0.2332      0.025     -9.443      0.000        -0.282    -0.185
==========================================================================================
```

By pivoting the predictions dataframe on trips_in_first_30_days and our categorical fields of interest, we can see how much our churn predictions change for different levels of each:

```
# create 2d plots of categories against trips_in_first_30
for name in ['city', 'phone', 'black_car_user']:
    cols = [c for c in DfSimPlot.columns if name in c] + ['trips_in_first_30_days']
    DfPivot = pd.pivot_table(data=DfSimPlot, values='y_pred', columns=cols, aggfunc=np.mean)
    u_range = range(len(DfOrig[name].unique()) - 1)
    DfPivot.unstack(u_range).plot(kind='line')
```







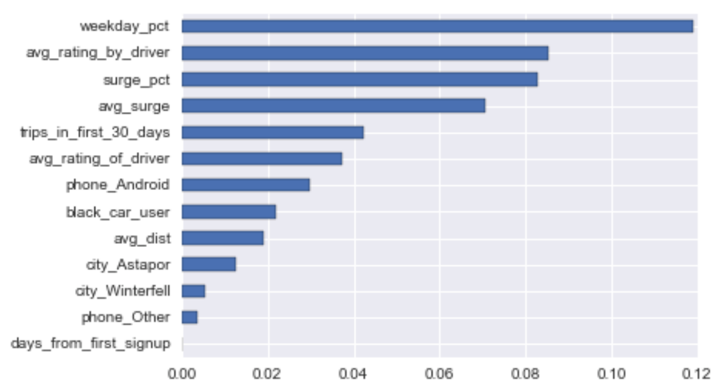
Appendix 1: Non-Linear Features

Mutual Information

Earlier in this exercise we looked at the direct relationship between each feature and churn by using ANOVA and Mutual Information. **ANOVA** captures the strength of the linear relationship between the two, whereas **Mutual Information** captures the non-linear relationship. While in our first pass we ignored the fact that some features had very high Mutual Information Scores but low ANOVA F-Statistics, we will now look into those other features to see if we can describe their relationship to churn.

```
# get mutual_info results
X_mc = DfDummies1.drop('churn', axis=1)
mc = mutual_info_classif(X_mc, y)

# print results
pd.Series(data=mc, index=X_mc.columns).sort_values().plot(kind='barh')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x15a7b6350>
```



Weekday Percent

Revisiting ANOVA with weekday_pct we find an F-Stat and p-value of 4.7 and 0.03, respectively:

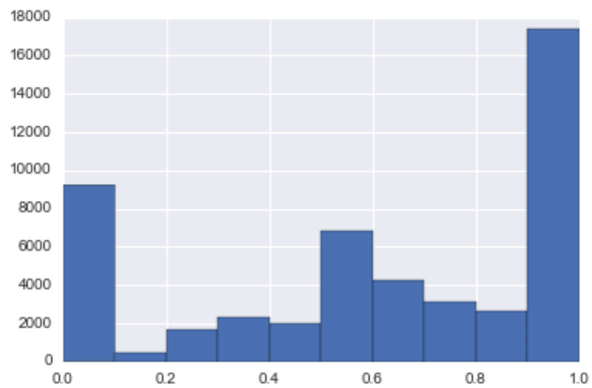
```
# updated model
cols = [c for c in X1.columns if 'weekday' in c]
fc = f_classif(X1[cols], y)

DfAnova = pd.DataFrame(np.vstack(fc).T, index=cols, columns=['F-Stat', 'p-value'])
print np.round(DfAnova.sort_values('F-Stat', ascending=False), 4)
```

```
             F-Stat  p-value
weekday_pct  4.6979   0.0302
```

However, the Mutual Information Score suggests there's a very strong non-linear relationship between weekday_pct and churn. Looking at the distribution of weekday_pct it's clear most values are 0 or 1.

```
X1['weekday_pct'].hist()
```

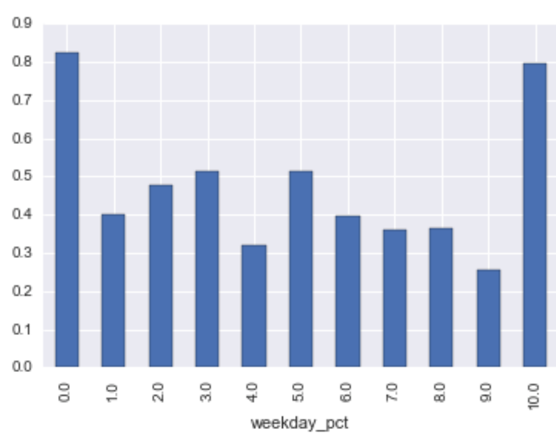
```
<matplotlib.axes._subplots.AxesSubplot at 0x14eabe790>
```



A comparison of churn at each level of of weekday_pct further supports that not only is the distribution different, but % churn for each group is different:

```
X1_temp = X1.copy()
X1_temp['churn'] = y
X1_temp.groupby(np.floor(X1['weekday_pct']/0.1)).mean()['churn'].plot(kind='bar')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1451f4bd0>
```



By transforming weekday_pct to a categorical variable, we might be able to get more information out of it. My hypothesis is that regular customers are rarely 100% weekday or weekend users, but that the customers who have 0 or 1 scores only took 1 weekday or weekend ride, ever, hence the higher churn rate:

```python
X2 = X1.copy()

# create conversion function
def convert_weekday_pct(x):
    if x == 0:
        return 'weekends'
    elif x == 1:
        return 'weekdays'
    else:
        return 'mix'

# get dummies
X2['weekday_pct'] = X2['weekday_pct'].apply(convert_weekday_pct).astype('category')
X2 = pd.get_dummies(X2)
X2 = X2.drop('weekday_pct_mix', axis=1)

# fit model
logit = sm.Logit(y, X2.drop(['avg_rating_of_driver'], axis=1))
result = logit.fit()
results_v2 = result.summary() # keep in back pocket
print results_v2
```

```
Optimization terminated successfully.
         Current function value: 0.509513
         Iterations 6
                           Logit Regression Results
==============================================================================
Dep. Variable:                      y   No. Observations:                50000
Model:                          Logit   Df Residuals:                    49987
Method:                           MLE   Df Model:                           12
Date:                Tue, 20 Dec 2016   Pseudo R-squ.:                  0.2305
Time:                        02:23:21   Log-Likelihood:                -25476.
converged:                       True   LL-Null:                       -33106.
                                        LLR p-value:                     0.000
==========================================================================================
                             coef    std err          z      P>|z|      [95.0% Conf. Int.]
------------------------------------------------------------------------------------------
avg_dist                   0.0186      0.002      8.424      0.000         0.014     0.023
avg_rating_by_driver       0.1409      0.025      5.610      0.000         0.092     0.190
black_car_user            -0.8290      0.023    -36.755      0.000        -0.873    -0.785
surge_pct                 -0.2185      0.060     -3.672      0.000        -0.335    -0.102
trips_in_first_30_days    -0.0403      0.003    -12.415      0.000        -0.047    -0.034
days_from_first_signup     0.0049      0.001      3.943      0.000         0.002     0.007
city_Astapor               1.7803      0.032     56.458      0.000         1.718     1.842
city_Winterfell            1.2735      0.029     44.189      0.000         1.217     1.330
phone_Android              1.0687      0.026     41.502      0.000         1.018     1.119
phone_Other                0.5682      0.123      4.632      0.000         0.328     0.809
intercept                 -2.0770      0.127    -16.380      0.000        -2.325    -1.828
weekday_pct_weekdays       1.5346      0.027     57.265      0.000         1.482     1.587
weekday_pct_weekends       1.6806      0.034     49.600      0.000         1.614     1.747
==========================================================================================
```

This hypothesis pays of huge. Not only are the F-statistics associated with this new dummy variable large:

```
# updated model
cols = [c for c in X2.columns if 'weekday' in c]
fc = f_classif(X2[cols], y)

DfAnova = pd.DataFrame(np.vstack(fc).T, index=cols, columns=['F-Stat', 'p-value'])
print np.round(DfAnova.sort_values('F-Stat', ascending=False), 4)
```

```
                        F-Stat  p-value
weekday_pct_weekdays  3340.8988      0.0
weekday_pct_weekends  2059.8616      0.0
```

The log-likelihood of the new model shows a big improvement:

```
# fit
logit = sm.Logit(y, X2.drop(['avg_rating_of_driver'], axis=1))
result = logit.fit()
results_v2 = result.summary() # keep in back pocket
print results_v2
```

```
Optimization terminated successfully.
         Current function value: 0.509513
         Iterations 6
                           Logit Regression Results
==============================================================================
Dep. Variable:                      y   No. Observations:                50000
Model:                          Logit   Df Residuals:                    49987
Method:                           MLE   Df Model:                           12
Date:                Tue, 20 Dec 2016   Pseudo R-squ.:                  0.2305
Time:                        02:39:01   Log-Likelihood:                -25476.
converged:                       True   LL-Null:                       -33106.
                                        LLR p-value:                     0.000
==========================================================================================
                             coef    std err          z      P>|z|      [95.0% Conf. Int.]
------------------------------------------------------------------------------------------
avg_dist                   0.0186      0.002      8.424      0.000         0.014     0.023
avg_rating_by_driver       0.1409      0.025      5.610      0.000         0.092     0.190
black_car_user            -0.8290      0.023    -36.755      0.000        -0.873    -0.785
surge_pct                 -0.2185      0.060     -3.672      0.000        -0.335    -0.102
trips_in_first_30_days    -0.0403      0.003    -12.415      0.000        -0.047    -0.034
days_from_first_signup     0.0049      0.001      3.943      0.000         0.002     0.007
city_Astapor               1.7803      0.032     56.458      0.000         1.718     1.842
city_Winterfell            1.2735      0.029     44.189      0.000         1.217     1.330
phone_Android              1.0687      0.026     41.502      0.000         1.018     1.119
phone_Other                0.5682      0.123      4.632      0.000         0.328     0.809
intercept                 -2.0770      0.127    -16.380      0.000        -2.325    -1.828
weekday_pct_weekdays       1.5346      0.027     57.265      0.000         1.482     1.587
weekday_pct_weekends       1.6806      0.034     49.600      0.000         1.614     1.747
==========================================================================================
```

And Accuracy and F1 take a big jump from 71.9% to 75.5% and 79.1% to 80.9%, because we have found a better way to describe the relationship between features and target through thorough, investigative feature engineering.

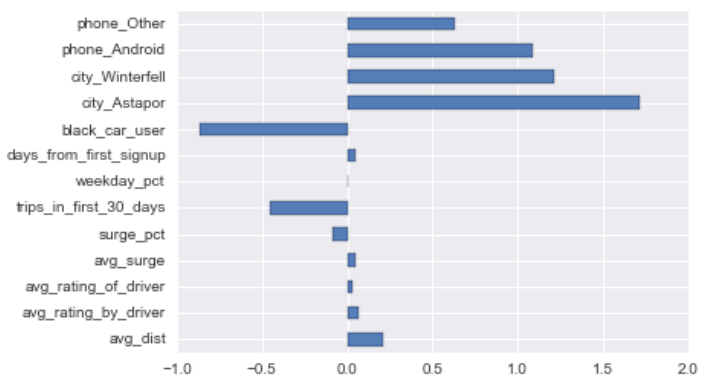
Appendix 2: Regularization

Full Model (Over-Fit Model)

Taking a look at model coefficients, we see every feature has a large effect, with binary variables having a particularly large one:

```
lrf = LogisticRegression(n_jobs=-1, penalty='l1', random_state=0, C=1000) # 1000 = no penalty
lrf.fit(X, y)
pd.Series(lrf.coef_[0], index=features).plot(kind='barh')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x10f542b50>
```



It's important to note that in the unpenalized model, every feature has "a say in the outcome." So every feature has a coefficient. It's also important to note that since the continuous features are on a larger scale than the binary ones, which are on a 0-1 scale, they have smaller coefficients.

Accuracy and F1 for this model are 71.8% and 79.1%, respectively. It's also very worth noting that precision (accuracy of guesses) is 74% whereas recall (% of target class observations you identify) is 85%. This will be important as we regularize.

```
print "Accuracy Score: {0:.3f}".format(metrics.accuracy_score(y_pred=y_pred, y_true=y))
print "F1 Score: {0:.3f}".format(metrics.f1_score(y_pred=y_pred, y_true=y))
print "Classification Report: \n\n {0}".format(metrics.classification_report(y_pred=y_pred, y_tr
```

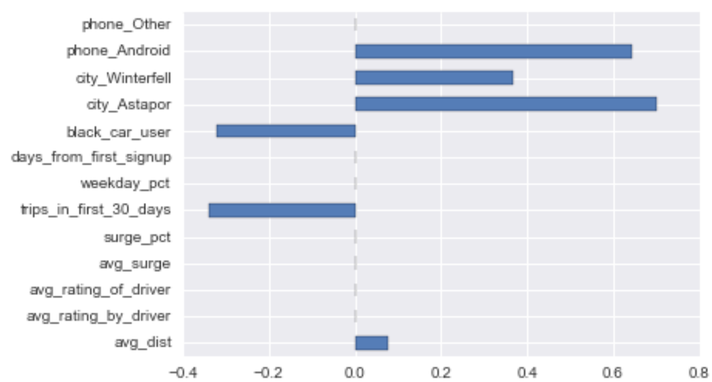
```
Accuracy Score: 0.718
F1 Score: 0.791
Classification Report:

             precision    recall  f1-score   support

          0       0.67      0.49      0.57     18804
          1       0.74      0.85      0.79     31196

avg / total       0.71      0.72      0.71     50000
```

L1 Regularization

The goal of the regularization parameter is to stop the model you're using from overfitting the data. L1 regularization in particular, aggressively penalizes coefficients such that some get suppressed to 0. In the above case we had a model where every feature had a coefficient and every coefficient was high. By penalizing the model for giving every feature a high coefficient, we can avoid overfitting, and can accomplish this by making the parameter C relatively low:

```
lr1 = LogisticRegression(n_jobs=-1, penalty='l1', random_state=0, C=0.001)
lr1.fit(X, y)
pd.Series(lr1.coef_[0], index=features).plot(kind='barh')
```

<matplotlib.axes._subplots.AxesSubplot at 0x10fd78690>



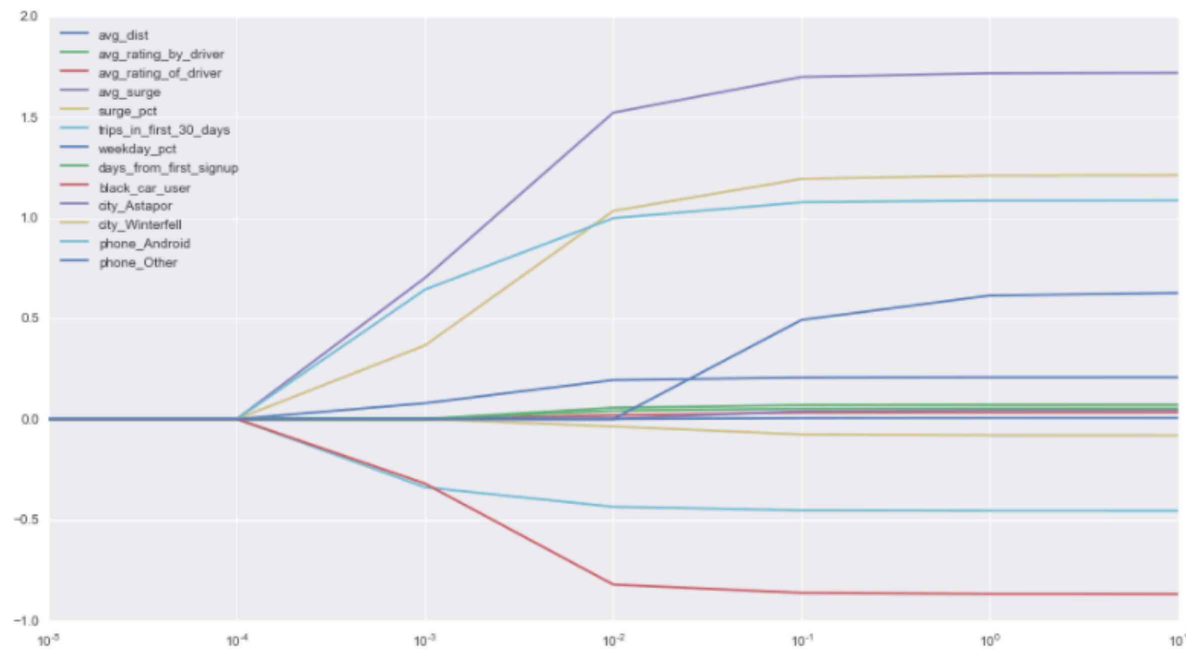
Settling on some value like C=0.001, we can see the factors that positively correlate with churn are: living in Astapor and Winterfell, having an Android phone, and taking longer distance rides. Features that negatively correlate are: using a black car and taking more trips in the first 30 days. Based on these findings alone, you can start to make suggestions on how to aggressively promote for certain cities, investigate the Android app experience, incentivize more trips in the first 30 days, or give discounts for longer distance rides… But back to regularization.

More generally, we can see the effect of changing the parameter C for each feature by iterating through C values:

```
c_range = np.arange(-5, 2)
coef_list = {}
for n in c_range:
    lr = LogisticRegression(n_jobs=-1, penalty='l1', random_state=0, C=10**n)
    lr.fit(X, y)
    coef_list[10**n] = lr.coef_[0]
```

```
df_coefs = pd.DataFrame.from_dict(coef_list)
df_coefs.index = features
df_coefs.T.plot(kind='line', figsize=(15, 8), logx=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1102b56d0>
```



In the above example, the regularization parameter C takes strong effect between 10^-4 and 10^-3, or 0.001 and 0.01.

Overall, this model is 1% less accurate than the overfit model, but has 0.5% better F1. Most notably though, precision fell to 70% and recall went up to 93%. How do you interpret that?

```
print "Accuracy Score: {0:.3f}".format(metrics.accuracy_score(y_pred=y_pred, y_true=y))
print "F1 Score: {0:.3f}".format(metrics.f1_score(y_pred=y_pred, y_true=y))
print "Classification Report: \n\n {0}".format(metrics.classification_report(y_pred=y_pred, y_tr
```

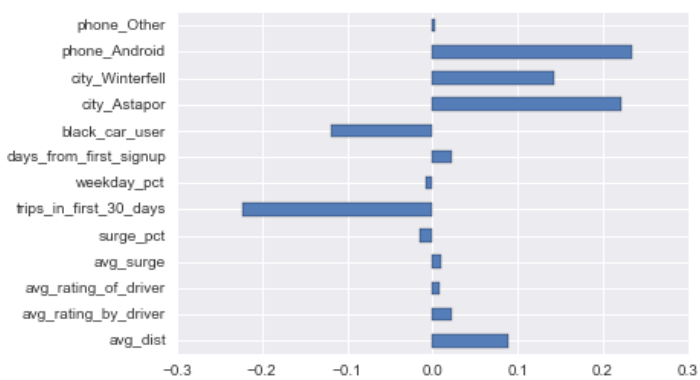
```
Accuracy Score: 0.706
F1 Score: 0.799
Classification Report: 

             precision    recall  f1-score   support

          0       0.75      0.33      0.46     18804
          1       0.70      0.93      0.80     31196

avg / total       0.72      0.71      0.67     50000
```

L2 Regularization

Similar to L1 Regularization, the L2 penalizes a model for overfitting, but does so more "smoothly." Instead of forcing some coefficients to 0, it just depresses all coefficients to some lower value simultaneously. You can see none of the coefficients have been zeroed out:

```
lr2 = LogisticRegression(n_jobs=-1, penalty='l2', random_state=0, C=0.0001)
lr2.fit(X, y)
pd.Series(lr2.coef_[0], index=features).plot(kind='barh')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x110b70890>
```

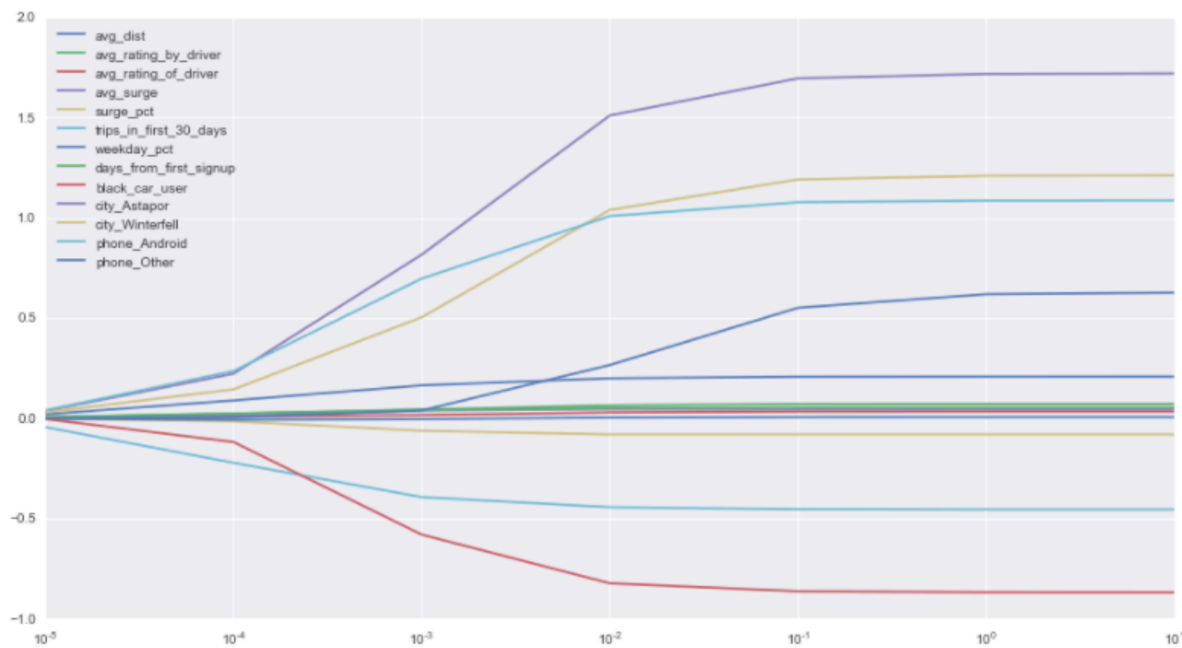


And that coefficients smoothly climb into their full-model values as penalization gets smaller:

```
c_range = np.arange(-5, 2)
coef_list = {}
for n in c_range:
    lr = LogisticRegression(n_jobs=-1, penalty='l2', random_state=0, C=10**n)
    lr.fit(X, y)
    coef_list[10**n] = lr.coef_[0]
```

```
df_coefs = pd.DataFrame.from_dict(coef_list)
df_coefs.index = features
df_coefs.T.plot(kind='line', figsize=(15, 8), logx=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x10f8c5cd0>
```



As with L1 Regularization, precision is lower and recall is higher than with the full model.

```
print "Accuracy Score: {0:.3f}".format(metrics.accuracy_score(y, y_pred))
print 'F1 Score: {0:.3f}'.format(metrics.f1_score(y, y_pred))
print "Classification Report: \n\n {0}".format(metrics.classification_report(y_true=y, y_pred=y_
```

```
Accuracy Score: 0.708
F1 Score: 0.797
Classification Report: 

             precision    recall  f1-score   support

          0       0.73      0.36      0.48     18804
          1       0.70      0.92      0.80     31196

avg / total       0.71      0.71      0.68     50000
```

Appendix 3: Bias/Variance

Out of Box Model

As before, running the model on a single test/train split of 20%, we find that the model is around 72% accurate between test and train.

```
from sklearn.cross_validation import train_test_split
from sklearn.linear_model import LogisticRegression

# get rf accuracy with out of box settings
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
lr = LogisticRegression(n_jobs=-1, random_state=0, penalty='l1', C=1000) # no penalty
lr.fit(X_train, y_train)
y_pred_lr = lr.predict(X_test)

print 'Training accuracy:', round(lr.score(X_train, y_train), 4)
print 'Test accuracy:', round(lr.score(X_test, y_test), 4)
```

```
Training accuracy: 0.7172
Test accuracy: 0.7231
```

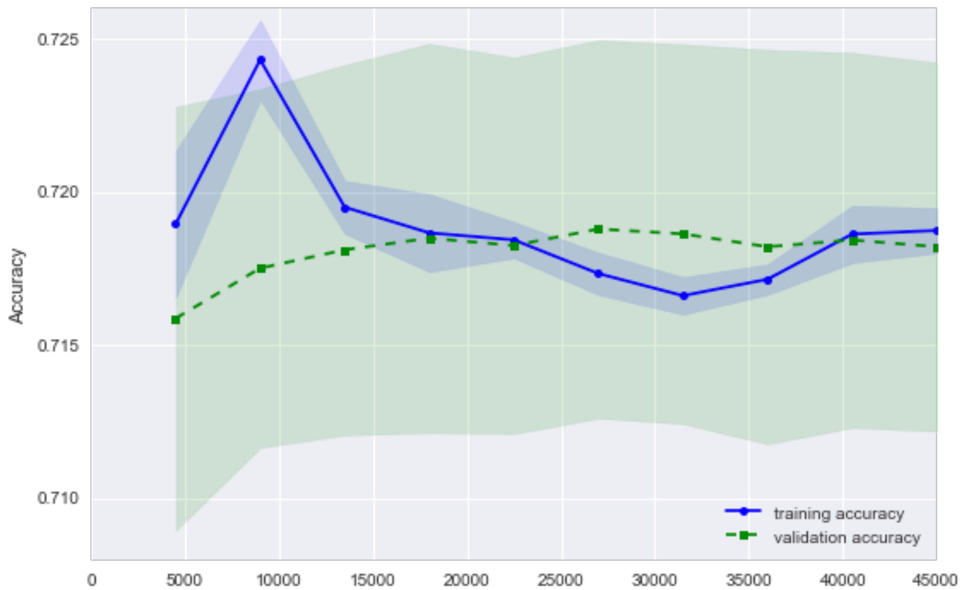
It also has an F1 of 79.5% and a very high recall—it captures 86% of all cases of churn:

```
from sklearn import metrics

# Accuracy, AUC, Precision, Recall, F1
print 'Key Metrics \n******************'
print 'Accuracy: %.3f' % metrics.accuracy_score(y_true=y_test, y_pred=y_pred_lr)
print 'F1 Score: %.3f' % metrics.f1_score(y_true=y_test, y_pred=y_pred_lr)
print 'Classification Report: \n\n {0}'.format(metrics.classification_report(y_test, y_pred_lr))
```

```
Key Metrics
******************
Accuracy: 0.723
F1 Score: 0.795
Classification Report:

             precision    recall  f1-score   support

          0       0.68      0.50      0.57      3749
          1       0.74      0.86      0.79      6251

avg / total       0.72      0.72      0.71     10000
```

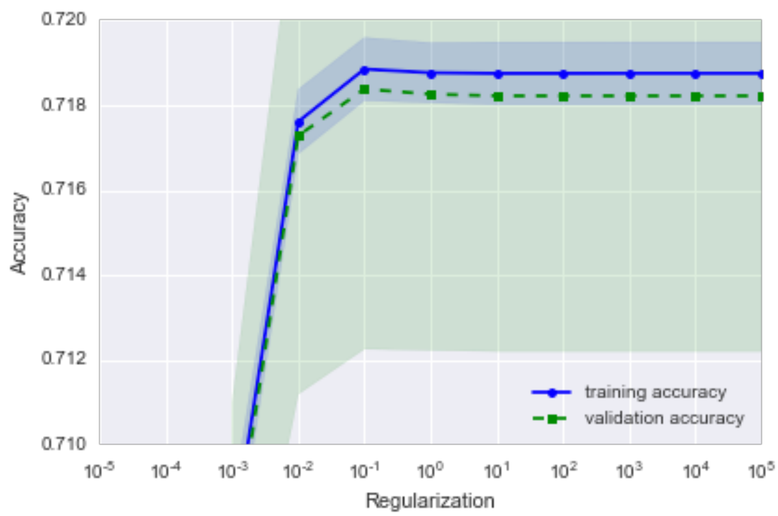
Learning Curves

Learning Curves are much more meaningful for a model like Logistic Regression because we expect test and training accuracy to be near each other. Versus Random Forest which explicitly over-fits on test data. Here we find that after 15k training examples, differences between train and test accuracy converge:



Validation Curves

Similarly, the validation curve hits peak accuracy at C=10^-1, and there isn't much difference in test/train for various levels of C. Meaning there's good bias-variance tradeoff:



Counter-Example

Grid search is good for finding optimal parameters, but it's not the best for accuracy. If seeking more accurate results, **the best thing you can do is find a better model or create an ensemble of models to represent your problem**. After you have found the best model, the next thing to focus on is feature engineering. To make the point, if we wanted an accurate model, we could have skipped Logistic Regression and used a Neural Network instead:

```
from sklearn.neural_network import MLPClassifier
from sklearn import metrics

mlp = MLPClassifier(activation='logistic')

# get rf accuracy with out of box settings
y_pred_mlp = cross_val_predict(estimator=mlp, X=X, y=y, cv=10)
```

```
# Accuracy, AUC, Precision, Recall, F1
print 'Key Metrics \n******************'
print 'Accuracy: %.3f' % metrics.accuracy_score(y_true=y, y_pred=y_pred_mlp)
print 'F1 Score: %.3f' % metrics.f1_score(y_true=y, y_pred=y_pred_mlp)
print 'Classification Report: \n\n {0}'.format(metrics.classification_report(y, y_pred_mlp))
```

```
Key Metrics
******************
Accuracy: 0.778
F1 Score: 0.829
Classification Report:

             precision    recall  f1-score   support

          0       0.74      0.64      0.69     18804
          1       0.80      0.86      0.83     31196

avg / total       0.78      0.78      0.78     50000
```

Accuracy is 5.9% better and F1 is 3.8% better, right out of the box.

Appendix 4: Feature Impact Analysis

[Defer to Regression Lesson]