HOMEWORK 1. ISYE 6501

Guillermo de la Hera Casado

August 22nd, 2019

Real life application of classification problems.

I work for a Pay-TV company with headquarters in Africa. Customers pre-pay monthly subscriptions and get channels specialized in different content e.g. football, movies, etc.

Our clients typically don't renew on time, due to complex transportation networks to payment points and inconvenience to do so during weekends. We identified the need of **predicting which customers are at risk of disconnection** in order to change their behavior with marketing initiatives.

We usually run Gradient Boosting Classifier as algorithm, probably it will come up later on in this course. Some of the predictors are as per below:

- Continuously active tenure (days)
- Activity within the last three months (days)
- Activity within the last six montnhs (days)
- Customer tenure since joining the product (days)
- Customer product ARPU ($)

SVM exercise

VanillaDot: finding a good classifier

The C parameter tells the SVM optimization how much we want to avoid misclassifying each training example.

- For large values of C, the optimization will choose a smaller-margin hyperplane if that hyperplane does a better job of getting all the training points classified correctly. There is a risk of overfitting.
- However, a small value of C may make optimizer to look for better generalization (larger-margin separating hyperplane), even if that hyperplane misclassifies more points.There is risk of underfitting.
- As per the homework guidelines, the algorithm should be trained with the full dataset and will not be validated with a test set, therefore generalization power is not a concern. We will check how increases in C affect both training accuracy.

In the following table, three measures are presented for each C value:

- *Overall accuracy:* (TP+TN)/(TP+TN+FP+FN)
- *% positive instances predicted correctly (recall):* TP / (TP+FN)
- *% negative instances predicted correctly (specificity):* TN / (TN+FP)

Table 1: Vanilladot. Training Accuracy per C value

C Value	Overall Accuracy	negative instances predicted correctly	positive instances predicted correctly
1e-04	54.7%	0%	100%
3e-04	55.7%	2%	100%
5e-04	65%	23.6%	99.2%
7e-04	80.7%	62.2%	96.1%
1e-03	83.8%	73.6%	92.2%
3e-03	86.4%	94.3%	79.9%
5e-03	86.4%	94.3%	79.9%

C Value	Overall Accuracy	negative instances predicted correctly	positive instances predicted correctly
1e-02	86.4%	94.3%	79.9%
1e-01	86.4%	94.3%	79.9%
1e+00	86.4%	94.3%	79.9%
1e+02	86.4%	94.3%	79.9%

By the look of it, very low values of C makes all the predictions be *yes's*. The consequence is that we are not able to predict the negative case, and therefore **we would be approving credit card applications for ALL customers** with the risk that implies.

C value = *3e-03* or *0.003* brings a modest raise in overall accuracy and a massive improvement on our capabilities to predict correctly *no's*, from 74% to 94%. For our business scenario it's very important to reduce False Positives to the very minimum, as we don't want to grant credit cards to customers with bad credit score. *C>0.003* values don't provide much more incremental improvement.

VanillaDot: finding the equation for the selected C Value: 0.003

Table 2: VanillaDot. Coefficients for C Value: 0.003

	Coefficients
A1	-0.0020000
A2	0.0320000
A3	0.0470000
A8	0.1110000
A9	0.3750000
A10	-0.2020000
A11	0.1700000
A12	-0.0050000
A14	-0.0250000
A15	0.0810000
a0	-0.2226155

[1] "The equation is: -0.002 A1 + 0.032 A2 + 0.047 A3 + 0.111 A8 + 0.375 A9 -0.202 A10 + 0.17 A11 -0.005 A12 -0.025 A14 + 0.081 A15 -0.222615544845752 = 0"

Comparing VanillaDot vs Radial Basis Function Kernel (RBF)

Table 3: RBF. Training Accuracy per C value

C Value	Overall Accuracy	negative instances predicted correctly	positive instances predicted correctly
1e-04	54.7%	0%	100%
3e-04	54.7%	0%	100%
5e-04	54.7%	0%	100%
7e-04	54.7%	0%	100%
1e-03	54.7%	0%	100%
3e-03	54.7%	0%	100%
5e-03	54.7%	0%	100%
1e-02	55.7%	2%	100%
1e-01	85.9%	94.3%	79.1%
1e+00	87.2%	94.3%	81.3%
1e+02	95.7%	94.3%	96.9%

C Value	Overall Accuracy	negative instances predicted correctly	positive instances predicted correctly

RBF behaves in a different way than VanillaDot. Only when C value gets to **0.1** the training accuracy improves considerably. Also, the higher C value is, the better RBF adapts to training set without any limitation. When C value is **100**, training accuracy is already at **96%, that is much higher than any of the VanillaDot outcomes.**

Something important to consider is that *sigma* parameter should be tuned as well for RBF, together with *C*. It hasn't be done in this example for simplicity. #matrix with nrows = dataset, ncol = k values, to store prediction values

K-nearest-neighbors exercise

For this exercise, both a main loop and a nested one have been created, in order to:

- Benchmark prediction accuracy for **different K values**. In this case, from k = 1 to k = 10.
- For each k value, **calculate the predicted value for each instance in the dataset**. That's done iteratively, through leave-one out cross validation. The predicted value will come as a continuous value. By using 0.5 as threshold, we follow this rule: *if it's >= 0.5, then it's transformed to 1. If it's <0.5, then it's transformed to 0.*
- For each k value, **calculate the overall prediction accuracy** across the dataset.

Table 4: KKNN. Training Accuracy per K value

K value	Overall Accuracy
1	81.5%
2	81.5%
3	81.5%
4	81.5%
5	85.2%
6	84.6%
7	84.7%
8	84.9%
9	84.7%
10	85%

As it can be seen in the above table **K value = 5** provides a great boost in accuracy: *[85.2%]* while still being a reasonable number. In addition, this is an odd number. As a learning doing some research, it seems that *odd numbers are recommended when doing binary classifications, since it eliminates the risk of getting ties e.g. two classes labels achieving the same score.* More info here, from Analytics Vidhya