Machine Learning Nanodegree - Project 02

Pedro M Duarte
pmd323@gmail.com
March 11, 2016

I. Submission Details

Below you will find descriptions of the files included in the project submission:

report.pdf: This report file.

01_student_intervention.ipynb / .html: Main IPython notebook where I filled all of the blanks marked as TODO.

02_explore_data.ipynb / .html: Bar charts for initial exploration of the data set.

03-exploringModels.ipyb / .html: In-depth exploration of the models and parameter tuning. Used to generate all of the plots that are included in the report.

II. Classification vs. Regression

Q:*Your goal is to identify students who might need early intervention - which type of supervised machine learning problem is this, classification or regression? Why?*

A: The possible outcomes in this case can be of only two categories: students need early intervention, or they do not. For that reason I identify this problem as a classification problem.

Strictly speaking, it is true that one may consider the probability that a student will pass their final exam or not, and with that granularity one could target more strongly students that are at the worst risk of failing. However, for the scope of this project I will not consider the actual pass/fail probability for each student.

III. Exploring the Data

Below we show answers to the questions from the guidelines:

Total number of students	395
Number of students who passed	265
Number of students who failed	130
Graduation rate of the class	67%
Number of features (excluding label)	30

IV. Preparing the Data

The data was prepared as indicated in the IPython notebook, by replacing 'yes' and 'no' by 1 and 0 respectively, and creating dummy variables to replace categorical variables.

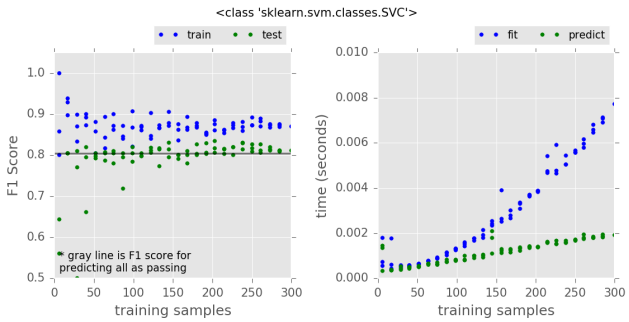

Fig. 1. Support Vector Classifier with sklearn default parameters. (Left) Learning curve. (Right) Train and predict time benchmarks.

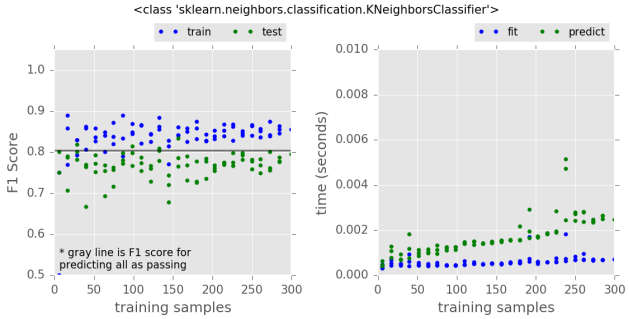

Fig. 2. KNeighbors Classifier (nn=5). (Left) Learning curve. (Right) Train and predict time benchmarks.

V. Training and Evaluating Models

For this section I implemented some code that allows me to easily plot the learning curve for a given sklearn model. The code can be seen in the attached IPython notebook titled 03-exploringModels.ipynb.

From the plots generated in that notebook I have selected the following models:

1) Support vector classifier.
2) Nearest neighbor classifier.
3) Decision tree classifier.

The learning curve and the time benchmark plots for training and prediction as a function of the number of training samples are shown in Figs. 1 to 3.

In what follows I discuss general points for each of the models that were selected. I will wait until the next section to provide the learning curves and performance tables, using tuned versions for each of the tree models chosen.

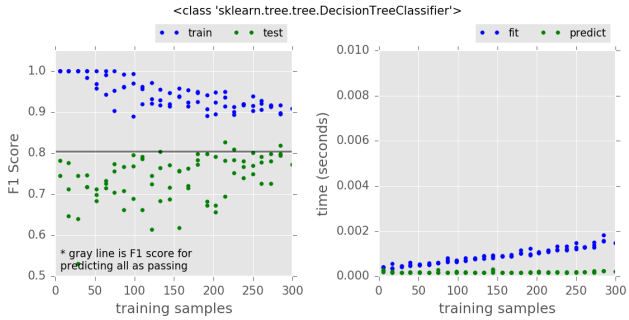

Fig. 3. Decision Tree Classifier (maxdepth=6). (Left) Learning curve. (Right) Train and predict time benchmarks.

A. Support Vector Classifier

The support vector classifier is most useful when the number of features in the data set is large. With a large number of features the classifier may be able to find decision boundaries that separate the pass and fail classes with the largest possible margin. It may be advantageous to the SVM that it does not have that many hyper-parameters to tune, so it may be easier to get a tuned model which does not overfit the data.

One possible disadvantage of the SVC is that it demands more resources for training, especially as the size of the training sample increases.

The performance of the SVC in terms of the F1 Score and train and predict times as a function of training sample size is shown in Fig. 1.

B. KNeighbors Classifier

The nearest neighbor classifier is interesting because training is very cheap. Essentially all it does is store all of the training data. The other side of that coin is that the model will have a slow time to make predictions, especially when the training set size is large. If the data is nicely clustered in the feature space, then KNN will make for a good classifier. The main parameter to adjust here is the number of nearest neighbors. Running a search on that parameter will allow us to hit the right level of bias/variance trade-off.

The performance of the KNN classifier in terms of the F1 Score, and the train and predict times as a function of training sample size are shown in Fig. 2.

C. Decision Tree Classifier

The decision tree classifier offers a lot of freedom in the parameters that you can tune. With enough depth, it can represent highly non-linear data, and for that reason it must be treated with a lot of care to avoid overfitting. Like the SVC, it is more time consuming to train the decision tree classifier, than it is to generate predictions from it. In comparison with the SVM, we observe that the decision tree classifier offers better training performance.

One disadvantage for the decision tree classifier might be the size of the training set that it needs to generalize over the problem. In that respect the SVM and KNN models may be superior.

The performance of the decision tree classifier in terms of the F1 Score, and the train and predict times as a function of training sample size are shown in Fig. 3.

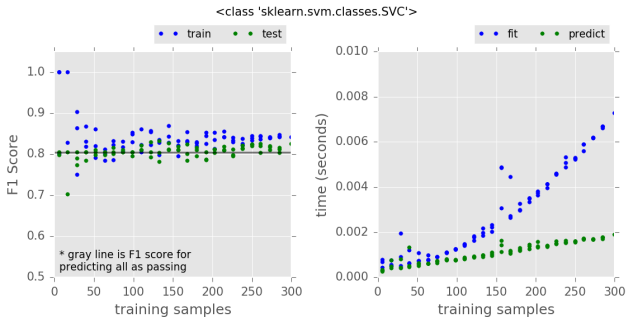

Fig. 4. Support Vector Classifier with tuned parameters. (Left) Learning curve. (Right) Train and predict time benchmarks.

VI. Tuning the model parameters

The code to generate the plots shown in this section can be found in the IPython notebook titled '03-exploringModels.ipynb'. The code used to generate the training size tables for F1 scores and timing performance can be found in '01-student-intervention.ipynb'.

A. Support vector classifier

For the support vector classifier we ran the following grid search:

kernel	'linear', 'rbf'
C	0.2, 1.0, 2.0
gamma	'auto', 0.1, 0.04, 0.02, 0.01, 0.008, 0.005

The best F1 score on the test set was 0.824, and was obtained by using the 'rbf' kernel and setting C=1.0 and gamma=0.01. The learning curve for the tuned model is shown in Fig. 4. In the figure, we observe that with the tuned parameters the model has less variance than with the default parameters.

size	train_time	predict_time	f1score_train	f1score_test
100	0.001937	0.001134	0.836601	0.797386
200	0.006036	0.003433	0.843658	0.802548
300	0.010464	0.003730	0.833333	0.805195

Fig. 5. Results for the SVC with 'rbf' kernel, C=1.0, and gamma=0.01. The F1 score is slightly different than the one reported in the main text due to the randomness of the train-test split used in each case.

size	train_time	predict_time	f1score_train	f1score_test
100	0.002566	0.013909	0.782609	0.810127
200	0.002431	0.007819	0.809816	0.815287
300	0.001159	0.006072	0.820833	0.812903

Fig. 7. Results for KNN with nneighbors=32. The F1 score is slightly different than the one reported in the main text due to the randomness of the train-test split used in each case.

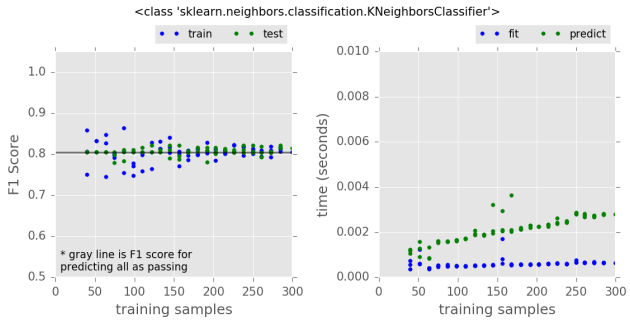

Fig. 6. KNeighbors Classifier with tuned parameters. (Left) Learning curve. (Right) Train and predict time benchmarks.

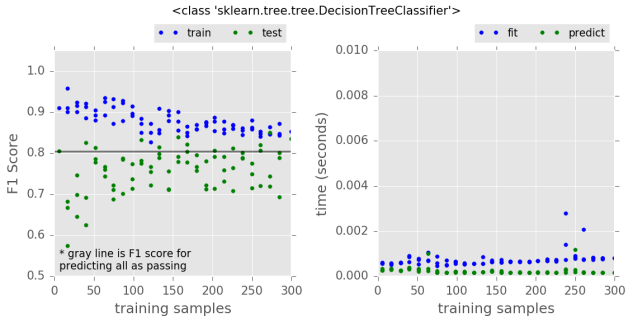

Fig. 8. Decision Tree Classifier (maxdepth=6). (Left) Learning curve. (Right) Train and predict time benchmarks.

The learning curve score shows a horizontal line that corresponds to the F1 Score that one would obtain if one were to predict all of the students as passing. In this case we see that we never do significantly better that this "all ones" prediction.

As we increase the size of the training set, we see only a slight improvement on the F1 score. The biggest effect of the training set size is on the training times, which are seen to increase substantially as we approach the full training set.

In Fig. 5, we show the results table for the training sizes specified in the rubric (all times are in seconds):

B. KNeighbors Classifier

For the KNN classifier we ran the following grid search:

nneighbors	1, 2, 4, 8, 16, 32, 64, 128, 150
weights	'uniform'

The best F1 score on the test set was 0.813, for nneighbors=32. The learning curve for these parameters is shown in Fig. 6. When running the grid search we noted that if we used too many neighbors, then the predictions became 'all passing', reflecting the imbalance of the data set in terms of passing and failing students.

The KNN model is always very cheap to fit, but the cost of predicting increases with the size of the training set. In terms of performance KNN is cheaper that SVC, but the F1 scores obtained are just slightly worse.

In terms of model variance, we see that both SVC and KNN do not have much variance. However, it is a little bit worrisome, for both SVC and KNN, that none of the two does much better than just predicting that all students will pass. In any case, we will argue that the ability to identify even a few students that have the highest need for intervention is a valuable output of the model.

The F1 scores and performance times for KNN are shown in Fig. 7.

C. Decision Tree Classifier

For the Decision Tree Classifier we ran the following grid search:

maxfeatures	12, 18, 24, 30
maxdepth	3, 4, 5, 6
minsamplessplit	2, 4, 8
minsampleslieaf	1, 2, 4

The best F1 score on the test set was 0.834, obtained for maxfeatures=30, maxdepth=4, minsamplessplit=8 and minsamplesleaf=1. The corresponding learning curve is shown in Fig. 8.

We observe here that the decision tree classifier exhibits some variance, and therefore it can be prone to overfitting. If we feed the decision tree with all of the training data, we may be just fitting the intricacies of the training set, which do not necessarily generalize over to the test set.

In terms of performance, we see good training and predicting times, in comparison with both SVC and KNN. Most importantly, the time performance does not seem to degrade significantly with training set size.

The results table for the decision tree classifier is shown in Fig. **??**.

VII. Choosing the best model

A. Explanation to the board of supervisors

Based on the results of tuning the parameters of the SVC, KNN and DecisionTree, I have selected the SVC as the best model.

In training we saw that the DecisionTree provided really good timing performance, however it was seen to be prone

size	train_time	predict_time	f1score_train	f1score_test
100	0.001149	0.000357	0.894410	0.818182
200	0.001343	0.000411	0.874576	0.800000
300	0.002917	0.001213	0.860412	0.811594

Fig. 9. Results for the Decision Tree with maxfeatures=30, maxdepth=4, minsamplessplit=8 and minsamplesleaf=1. The F1 score is slightly different than the one reported in the main text due to the randomness of the train-test split used in each case.

to large variance. Some good F1 scores were observed in the DecisionTree, but those may have been a statistical anomaly, since the model exhibited considerable variance.

The KNN model has good F1 scores, but the problem with it is the performance when making predictions.

The SVC takes slightly longer to train than both KNN and DecisionTree if we use the entire training set, but we also saw that it is not necessary to use the entire training set to obtain robust results with the SVC. The learning curve in Fig. 4 shows us that with greater than approximately 100 samples we already obtain results that have F1 score that is comparable with what we get if we use the entire set of training data.

By selecting the SVC we obtain a robust model, which we think will perform equally well for future realizations of the data. If we have sufficient resources we can train the model using all of the data available, however if the school runs into issues with computing power, we can safely dial down the amount of data used in training and still obtain reliable results with good prediction times.

I believe that the SVC offers the best advantages among the three models studied.

B. How does SVC work in layman terms?

The support vector classifier works by drawing imaginary boundaries between groups of data points that belong to the same class.

The assumption we make is that the values of the data features tend to be clustered together for passing and failing students. A support vector machine is an algorithm that allows us to go over the training data and determine the mathematical boundary that most effectively splits the samples into clusters of passing and failing students.

In its simplest incarnation, the support vector classifier restricts itself to the use of mathematical boundaries that are linear. For example, as shown in Fig. 10, we could have a data set of passing and failing students that can be optimally separated by a straight line in the 2-dimensional space defined by Feature A and Feature B in the data set.

In the general case the support vector classifier can make use of a trick, called the kernel trick, which allows it to come up with decision boundaries that are not necessarily linear in the feature space. An example is shown in Fig. 11, where the kernel trick allows the support vector classifier to find and appropriate boundary to optimally divide the passing and failing samples.

After training is complete, we know the exact mathematical form of the boundary that was determined by the support

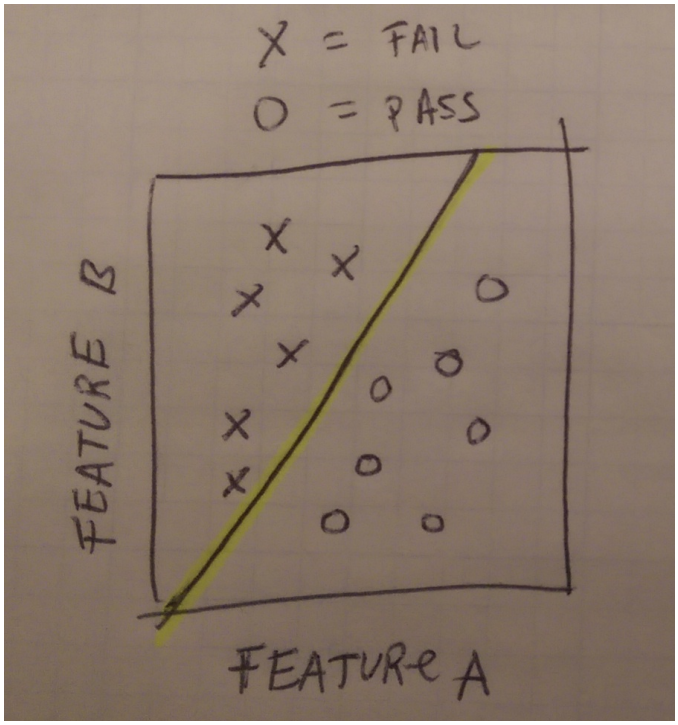

Fig. 10. Support vector classifier with linear boundary.

vector classifier for our training data. To make a prediction, we simply find out on which side of the dividing boundary does the student in question fall into. If the student falls into the passing side of the boundary, we predict that the student will pass the exam, and viceversa if the student falls into the failing side of the boundary.

Support vector classifiers are a robust model that works well when we have a large number of features in our data. As we observed in our analysis, we do not need to have a very large number of training observations to reliably train a support vector classifier.

VIII. Fine-tune the model

To fine tune the SVC model we run a GridSearchCV in sklearn. The parameter grid is chosen as follows:

kernel	'linear', 'rbf'
C	0.2, 1.0, 2.0
gamma	'auto', 0.1, 0.04, 0.02, 0.01, 0.008, 0.005

The gridsearch results are shown in the table below and sorted by the F1 score on the test set. For this run, **the best**

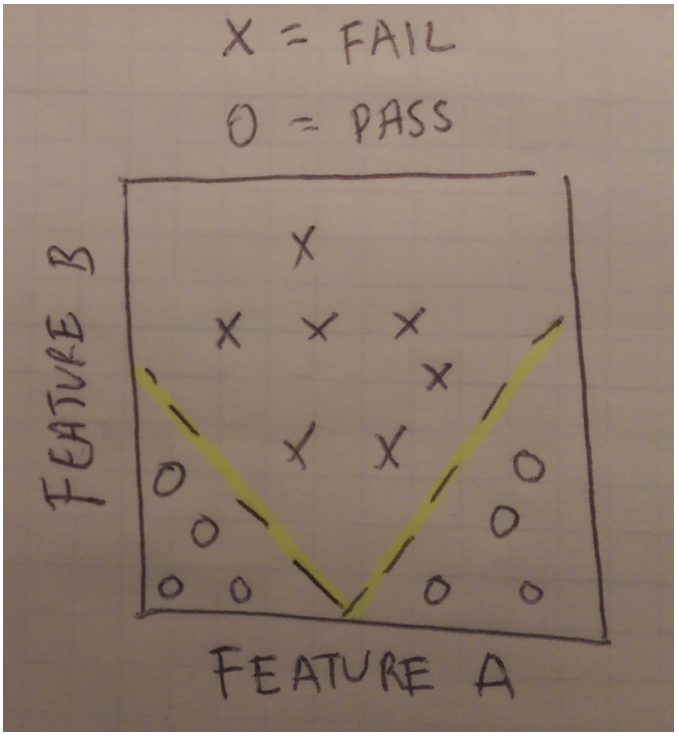

Fig. 11. Support vector classifier with non-linear boundary.

F1 score on the test set was 0.834, and was obtained using the 'rbf' kernel, with C=1.0 and gamma=0.1.

REFERENCES

[1] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

kernel	C	gamma	cv_score	test_score
rbf	1.0	0.1	0.807337	0.820513
rbf	2.0	0.005	0.809009	0.810458
rbf	1.0	0.008	0.804864	0.805195
rbf	1.0	0.01	0.808145	0.805195
rbf	0.2	0.1	0.802395	0.805031
rbf	0.2	0.04	0.802395	0.805031
rbf	0.2	0.02	0.802395	0.805031
rbf	0.2	0.008	0.802395	0.805031
rbf	0.2	0.005	0.802395	0.805031
rbf	0.2	auto	0.802395	0.805031
rbf	0.2	0.01	0.802395	0.805031
rbf	1.0	0.005	0.807248	0.802548
rbf	2.0	0.008	0.808632	0.800000
rbf	2.0	0.01	0.809682	0.800000
rbf	2.0	0.02	0.812386	0.800000
rbf	2.0	auto	0.807083	0.800000
rbf	2.0	0.1	0.790486	0.794702
rbf	1.0	0.04	0.806758	0.794702
rbf	1.0	0.02	0.805043	0.794702
rbf	1.0	auto	0.807585	0.794702
rbf	2.0	0.04	0.790085	0.791667
linear	2.0	0.1	0.751998	0.791367
linear	2.0	0.04	0.751998	0.791367
linear	2.0	0.005	0.751998	0.791367
linear	2.0	0.008	0.751998	0.791367
linear	2.0	0.01	0.751998	0.791367
linear	2.0	auto	0.751998	0.791367
linear	2.0	0.02	0.751998	0.791367
linear	0.2	0.1	0.788244	0.780142
linear	0.2	0.04	0.788244	0.780142
linear	0.2	0.02	0.788244	0.780142
linear	0.2	auto	0.788244	0.780142
linear	0.2	0.01	0.788244	0.780142
linear	0.2	0.008	0.788244	0.780142
linear	0.2	0.005	0.788244	0.780142
linear	1.0	0.02	0.759963	0.776978
linear	1.0	0.005	0.759963	0.776978
linear	1.0	auto	0.759963	0.776978
linear	1.0	0.008	0.759963	0.776978
linear	1.0	0.01	0.759963	0.776978
linear	1.0	0.1	0.759963	0.776978
linear	1.0	0.04	0.759963	0.776978

TABLE I
GRIDSEARCH RESULTS.