Project Report: Student Intervention System

Aravind Battaje

April 9, 2016

1 Project Steps

The goal of the project is to find a suitable supervised machine learning algorithm that can identify students who might need early intervention, by predicting their success/failure in graduation from the current records. To accomplish this task, three supervised learning models are probed for their suitability, and the best of them is later fine-tuned to get the best possible learning algorithm. All the three models are realized with the help of `scikit-learn`. For each of three models, a preliminary analysis is performed about its suitable basic configuration, costs and performance. Because of the small size of the dataset, cost and performance are measured by running multiple trials on shuffled data. The overall best performer that balances cost and performance is then fine-tuned using grid search (on parameters) and final predictions are made.

2 Classification vs Regression

A machine learning algorithm can be classified into two types, based on its nature of outputs, viz., classification and regression. Classification supports outputs of discrete values and regression outputs continuous values. This project entails a classification type of problem because the output desired from the *intervention system* is discrete in nature, i.e., a student graduates or not from his/her current characteristics. Regression would be more suitable for, say an algorithm that predicts the final exam score from a student's current academic records.

3 Dataset

Several qualities of students such as their family background, social characteristics, extra-curricular activities, etc., along with the information if they graduated or not, are given along with the project in `student-data.csv`. The dataset possesses following characteristics:

Total number of students	395
Number of students who passed	265
Number of students who failed	130
Graduation rate of the class	67.09%
Number of features of dataset	30

4 Training and Evaluating Models

Three supervised learning algorithms from `scikit-learn` were probed for their potential in *best* modeling the student intervention problem.

4.1 Naive Bayes Classifier

Naive Bayes Classifier is one of the simplest, and yet powerful algorithms used in supervised learning. It is a subset of the more complex generalization – Bayesian inference – in that, the likelihood of a hypothesis

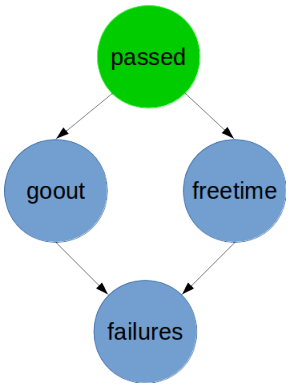
for modeling the data is inferred from the likelihood of data given a hypothesis. A contrived example which uses some of the features of the data provided in this project (`student-data.csv`) illustrates a few basic principles. Figure 1a shows a belief network modeled with the classification variable *passed* dependent on three features, with a certain conditionality evident between them. From the data (training), conditional probability tables for each node is calculated, and so when a new data sample (testing) is introduced, the joint probability for *passed* is calculated by simply multiplying relevant entries from the table. That is, assuming v_1, v_2 and v_3 represent the variables from the sample for *goout*, *freetime* and *failures* respectively,

$$P(passed = 1, goout = v_1, freetime = v_2, failures = v_3)$$

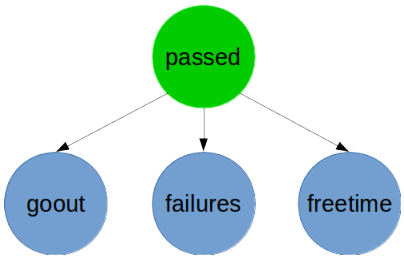
will give the probabilty of student passing. This joint probability can be expressed as,

$$P(...) = \prod_{i=1}^{n} P(v_i | Parents(Y_i)),$$

where $P(passed = 1)$, $P(goout = v_1 | passed = 1)$, $P(freetime = v_2 | passed = 1)$ and $P(failures = v_3 | freetime = v_2, goout = v_1)$ will be the individual terms, derived from the belief network.



(a) Bayesian Inference



(b) Naive Bayes

Figure 1: Example belief networks with some of the features of `student-data`. *passed* is the classification variable indicating if a student will pass or not; *goout* indicates the time spent by a student going out with friends; *freetime* indicates the free time a student gets after school; and, *failures* indicates a student's failures in the past.

Naive Bayes Classifier shown in Figure 1b on the other hand assumes conditional independence between the features, and models the classification variables as being directly dependent on all features. The joint probability $P(passed = 1, goout = v_1, freetime = v_2, failures = v_3)$ thus is much simpler:

$$P(...) = P(passed = 1)P(goout = v_1 | passed = 1)P(freetime = v_2 | passed = 1)P(failures = v_3 | passed = 1)$$

Computationally this is cheaper than the more complicated belief network of Figure 1a, but more importantly, the computational cost of finding the best arrangement of nodes in the belief network during training grows exponentially with number of nodes (or features). Therefore, Bayesian inference – ideally considered as the *gold* standard for supervised learning – becomes impractical.

Nevertheless, Naive Bayes still stands out pretty well even as it assumes no causality between the several features of the data. It has been shown to work surprisingly well in many applications, especially in spam-filtering and such text-based classifiers [MAP06]. Although the dataset provided along with the project doesn't qualify as having much text based data, and it could sway more towards a dataset which has correlated data features, such as in Figure 1a, it has been shown that even with the independence assumption on dependent features, as in Figure 1b, Naive Bayes works pretty well [Ris01]. Naive Bayes classifier is also extremely computationally efficient for both training and predicting, considering the amount of calculations involved and it could ideally set a benchmark for the other algorithms to compete against. However, the

	Training set size		
	100	200	300
Training time (msec)	1.234055	1.497984	1.840115
Prediction time - Training set (msec)	0.578880	0.742912	0.931025
Prediction time - Testing set (msec)	0.527859	0.520945	0.528097
F1 score - Training set	0.601942	0.770428	0.799031
F1 score - Testing set	0.489362	0.730159	0.800000

Table 1: Performance of Naive Bayes Classifier (single run)

	Training set size		
	100	200	300
Training time (msec)	1.140807	1.380339	1.651909
Prediction time - Training set (msec)	0.541310	0.731080	0.921652
Prediction time - Testing set (msec)	0.528181	0.530350	0.533614
F1 score - Training set	0.696105	0.796343	0.797403
F1 score - Testing set	0.599165	0.725537	0.753784

Table 2: Performance of Naive Bayes Classifier (100 runs)

downside is that the dataset size is quite small considering the number of features, and so Naive Bayes is susceptible to the *curse of dimensionality*.

`scikit-learn`'s `GaussianNB()` is used to train 300 samples from the given dataset in default setting; no additional parameters are provided. A single run followed by 100 runs (Refer 6.1) is performed and the corresponding results are logged in Table 1 and Table 2 respectively. It can be seen that the training times for the classifier is in the order of a millisecond, and it seems to grow linearly with the training set size. The same holds true for prediction times, and the F_1 scores on the test data for a training set size of 300 is remarkable, given the computational cost.

4.2 Support Vector Machine

Support Vector Machines (SVM) is one of the most ubiquitous supervised learning algorithms in the realm of statistical inference and machine learning. It is exemplary of a class of classifiers that try to maximize the *margin* for a decision boundary between two classes [BGV92]. Data (or model) generalization – at least to some extent – is thus inherently a part of SVM, as the maximum margin ensures that the decision boundary is not too close to the data points, which should prevent over-fitting. It is also incredibly economical as computing of maximum margin and the decision boundaries really boils down to dot products of data sample vectors in the former and data sample vector with sample to predict for the latter. In its simplest form, SVM based classifier (SVC) linearly separates data points with the maximum margin hyper-plane. To work on non-linearly separable data, SVM algorithm also makes it very easy to use *kernel trick* in the model; the previously noted dot product advantage makes the application of a kernel on the data have barely any computational overhead.

Figure 2 shows a contrived example that considers two non-linearly separable features from the given dataset. It illustrates the extension of the nascent form of SVC (linear separator) with a simple radial basis function (RBF) kernel. It can be seen that the RBF kernel allows formation of complex decision boundaries within the rules of maximum *soft* margin, and fits the data pretty well.

SVC has been proven to work extremely well for text recognition and categorization tasks [BGV92, Joa98], such as in optical character readers. It indicates the nature of SVC to adapt to highly non-linear and inter-dependent data features. In the dataset given, it can be said that some of the features are interdependent to an extent, and more importantly are highly non-linear in nature, with an exaggerated sense of discreteness (2 or 3 values). Additionally, SVMs are very amenable to non-linear score metrics such as F_1 score used in the project [Joa05]. Thus SVC should ideally suit the student intervention system context naturally.

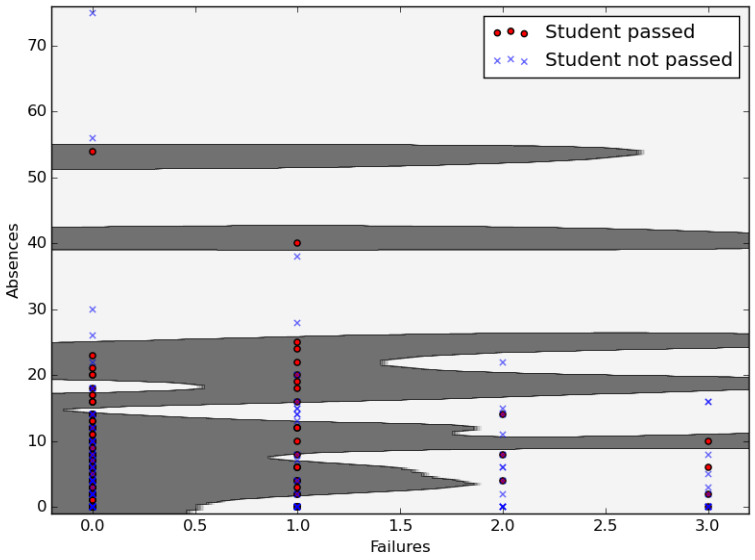


Figure 2: Example Support Vector Machine based Classifier with an RBF kernel; *Failures* indicates a student's past failures; *Absences* indicates number of classes a student was absent for. Decision boundaries in grey (student passed) and white (student not passed) are underlying the data points of *passed* label from the dataset.

	Training set size		
	100	200	300
Training time (msec)	1.495123	4.133940	11.901855
Prediction time - Training set (msec)	0.791788	2.635002	6.358147
Prediction time - Testing set (msec)	0.774860	2.138138	2.794027
F1 score - Training set	0.923077	0.895425	0.891353
F1 score - Testing set	0.805195	0.812903	0.818182

Table 3: Performance of SVC Polynomial 2^{nd} degree Kernel (single run)

	Training set size		
	100	200	300
Training time (msec)	2.232075	4.521847	9.538889
Prediction time - Training set (msec)	1.032114	3.026009	6.134987
Prediction time - Testing set (msec)	0.784874	1.445055	2.069950
F1 score - Training set	0.917197	0.907285	0.905405
F1 score - Testing set	0.820513	0.812903	0.818182

Table 4: Performance of SVC Polynomial 3^{rd} degree Kernel (single run)

`scikit-learn` provides `SVC()` that is capable of working with several types of kernels and parameters. Single runs – training and testing – with `SVC()` having *linear*, *rbf*, *poly* (*degree* 2 and 3), and *sigmoid* kernels were performed. The (rough) performance extracted from the single runs are then used to decide on the *best* kernel configuration for the context. For all the runs, *best practices* for SVMs [HCL+03] were followed, such as scaling input data values ($[-1, +1]$) so that the features carry zero mean and unit variance. Before scaling, the dataset was shuffled and split into 300 data points for training (later used in subsets) and the rest for testing. The same scaling transformation used for training set was used to scale the test set at the time of `predict`. For the single runs, only the performance data pertaining to good contentendors (kernels)

	Training set size		
	100	200	300
Training time (msec)	1.450677	4.090371	8.153820
Prediction time - Training set (msec)	0.807276	2.591932	5.424671
Prediction time - Testing set (msec)	0.760419	1.305692	1.772885
F1 score - Training set	0.915101	0.903190	0.896303
F1 score - Testing set	0.799621	0.796329	0.798391

Table 5: Performance of SVC Polynomial 2^{nd} degree Kernel (100 runs)

	Training set size		
	100	200	300
Training time (msec)	1.749039	5.638123	11.893034
Prediction time - Training set (msec)	1.103163	4.788876	8.475065
Prediction time - Testing set (msec)	1.050949	2.451181	2.701044
F1 score - Training set	0.929032	0.901316	0.895323
F1 score - Testing set	0.807692	0.802548	0.818182

Table 6: Performance of SVC RBF Kernel (single run)

	Training set size		
	100	200	300
Training time (msec)	1.721036	5.187237	10.721319
Prediction time - Training set (msec)	1.109087	3.853130	8.189697
Prediction time - Testing set (msec)	1.050169	1.879785	2.673924
F1 score - Training set	0.931261	0.915022	0.905248
F1 score - Testing set	0.801439	0.805833	0.808297

Table 7: Performance of SVC RBF Kernel (100 runs)

are provided below. *linear* kernel was found to be fast, but performed on par with Naive Bayes classifier in terms of F_1 score. *sigmoid* kernel did not suit the dataset very well; it didn't make any perceptible decision boundary and was independent of the training set size. Two kernels, viz., *rbf* in Table 6 and *poly* (*degree* 2) in Table 3 were found to perform well on the dataset. *poly* (*degree* 3) in Table 4, also seems to be doing pretty good, but it was found that over-fitting was imminent (as evident in the slightly higher testing F_1 score here) and also the score tended to vary a lot with different dataset splits.

100 runs are performed with the best kernels; Table 5 records the performance of *poly* (*degree* 2) kernel, and Table 7 records that of *rbf* kernel. With 100 runs, similarities in their performance becomes prevalent; F_1 scores on the test data set seem to be very close to each other. Objectively *poly* kernel seems to be better than *rbf* because of the lesser training and prediction times, but given the robustness of *rbf* in other application domains [HCL+03], it wins out against *poly*.

4.3 Boosting

Boosting is a class of ensemble-based supervised learning (meta-)algorithms that follow the principle of assembling a *strong* estimator from a bunch of *weak* estimators. By definition, a *weak* estimator has a performance only better than predicting by chance, i.e., error of lesser than 0.5 in binary classification. Boosting iteratively tries to fit these *weak* estimators to the data by exaggerating weights of the samples that were misclassified in the previous iteration. This effectively changes the *distribution* of data samples from the perspective of the *weak* learners or hypotheses. The final hypothesis then will be similar to a weighted sum of the individual hypotheses gathered in each iteration. The weights are a function of the prediction

error in the iteration. This simple technique of adapting weak estimators to form a voting (vaguely) ensemble is very powerful [Sch90], and a robust, well-proven version of it, called *AdaBoost* [FS97] is used here. Quite surprisingly, it has been shown that AdaBoost (and boosting in general) is highly resistant to overfitting [Sch13], given truly *weak* estimators are used, sufficient margins are created in each iteration and data is not too noisy. Thus, Boosting could ideally be the best algorithm for this project. The downside however, could be the (linearly) scaling computational cost, which gets added up from the individual estimators.

	Training set size		
	100	150	200
Training time (msec)	117.190123	142.978907	119.709969
Prediction time - Training set (msec)	9.433985	9.639978	10.140896
Prediction time - Testing set (msec)	11.713028	7.894993	7.919073
F1 score - Training set	0.937500	0.855072	0.827586
F1 score - Testing set	0.509091	0.581818	0.612903

Table 8: Performance of AdaBoost Classifier (single run) with balanced dataset

	Training set size		
	100	150	200
Training time (msec)	116.995811	116.348028	116.556168
Prediction time - Training set (msec)	8.880124	9.707942	10.348394
Prediction time - Testing set (msec)	8.139987	8.327084	8.233488
F1 score - Training set	0.951172	0.864298	0.819485
F1 score - Testing set	0.586345	0.614715	0.622481

Table 9: Performance of AdaBoost Classifier (100 runs) with balanced dataset

	Training set size		
	100	200	300
Training time (msec)	137.079000	130.676985	129.156113
Prediction time - Training set (msec)	8.861065	10.184050	11.499882
Prediction time - Testing set (msec)	8.553982	8.610010	8.504868
F1 score - Training set	0.993103	0.901408	0.865882
F1 score - Testing set	0.753623	0.788732	0.772414

Table 10: Performance of AdaBoost Classifier (single run) with identical dataset as used for other models

	Training set size		
	100	200	300
Training time (msec)	108.652627	116.185341	125.313888
Prediction time - Training set (msec)	8.656120	10.007663	11.349833
Prediction time - Testing set (msec)	8.542781	8.599198	8.634429
F1 score - Training set	0.975519	0.885755	0.859142
F1 score - Testing set	0.738435	0.764309	0.770984

Table 11: Performance of AdaBoost Classifier (100 runs) with identical dataset as used for other models

`AdaBoostClassifier()` from `scikit-learn` makes it possible to run any *weak* base estimator that supports sample weighting. In the project, `DecisionTreeClassifier()` is used as base estimator. Since

decision trees have a natural tendency to bias towards the dominant class label, and the dataset provided along with the project has more `'passed' == 'yes'` labels, *data balancing* was performed. This step entails eliminating entries in the dataset that pertain to the dominant label, to equalize with number of non-dominant ones. The number of samples was thus reduced to 260 from the 365 of the unbalanced dataset. So the shuffle-split procedure results in 200 training and 60 test samples, and the training set sizes becomes 100, 150 and 200. Table 9 shows this difference in training set sizes, in comparision to previous algorithms such as in Table 2.

With the above mentioned data balancing setup, sample runs with two configurations are performed. First, a strong base estimator, i.e., a decision tree of `max_depth` 15 is used. It was observed that training F_1 scores were always 1.0 for all training sizes. This shows the potential of overfitting with AdaBoost due to non-adherence of the weak estimator clause. The second (practical configuration) uses a weak decision tree (stump) with `max_depth` 1 as its base estimator. A single run (Table 8) with this configuration is verified to be better than first, and so performance is measured by averaging 100 runs. This is shown in Table 9. It is apparent from the tables that the training times is almost an order of magnitude more than the previous models, but it remains more or less constant with different training set sizes because AdaBoost stops at a specified number of iterations (`n_estimators`), and doesn't directly depend on the input size. More importantly, it can be seen that the F_1 scores seem to be much lesser than the previous models on test set. This is only as a consequence of *data balancing*, and so *direct* comparison of models is impossible. So, to obtain equivalent scores, a single run and 100 runs with the same dataset configuration as used with other models were also performed and are logged in Table 10 and Table 11 respectively. In case indirect comparison is needed, Section 6.2.3 discusses a possible mechanism. Using the performance results directly from Table 11, it can be concluded that with the given dataset, AdaBoost doesn't perform very well, and the F_1 scores are on par with Naive Bayes model. This could change however with a larger, and preferably already balanced, dataset.

5 Finding the Best Model

Out of the three models tried above, Support Vector Machine based classifier (SVC) that uses a radial basis function (RBF) kernel seems to be the best model.

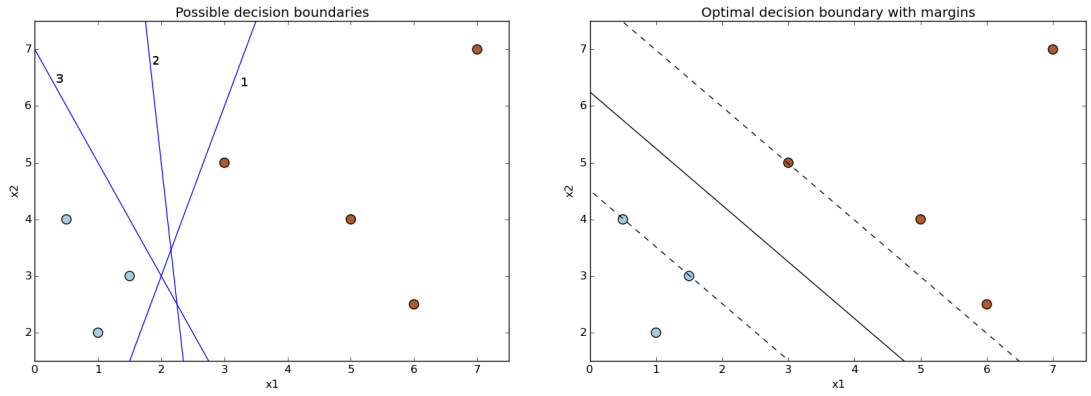
Objectively comparing the performance tables above, it can be seen that Naive Bayes classifier (Table 2) is a very good contender. Its computational efficieny – just 1.65 ms training time with 300 data points – is unmatched, and F_1 score appears to grow with the training size. It cannot be immediately concluded from the table that the F_1 score could continously improve with training set size. However, it should be apparent – at least from the *simplicity* of the model – that it cannot differentiate correlated data (features that assume causality). And given student data appears to have a few features that could be highly correlated, the *assumptions* made in the model might give in. There are also no *parameters* to tune with Naive Bayes classifier. This is really not a qualification criteria for a model, but it could be helpful if there would have been some parameter that could adapt the classifer to a specific context. Boosting classifier (AdaBoost) was very expensive computationally – around 120 ms – as shown in Table 11, and somehow, it didn't fit very well in the context of the given dataset. As already pointed out, the F_1 scores cannot be directly compared to other models if dataset balancing is used, but an equivalent F_1 score could be roughly derived if comparison is inevitable in Section 6.2.3. Considering the performance of unbalanced dataset, AdaBoost's F_1 score is more or less similar to that Naive Bayes. However, if a larger dataset (maybe 10 times larger) were given, AdaBoost could be re-evaluated and it might show much better results. It is assumed however that a larger dataset wouldn't be available.

SVC in a sense, balances both ends of the spectrum covered by the other two models. It doesn't intrinsically make naive assumptions on the dataset as Naive Bayes Classifier, and it doesn't rely on intense computations such as with Boosting classifier. Although SVC is not completely resistant to the *curse of dimensionality*, it gives an advantage over relatively smaller datasets as it inherently tends towards data generalization. For the given dataset, it can be seen from Table 7 (and Table 5) that SVC achieves both the best F_1 scores (around 0.80) and the best balance of computational efficiency and model accuracy. As already noted, polynomial kernel is a very close competitor to the RBF kernel based SVC, and RBF is chosen as most appropriate both because of its marginal improvement in F_1 score and because of its perceived robustness.

Also, like for other models, SVC would definitely benefit from a larger dataset to improve accuracy, and since it appears that training times almost linearly grows with the dataset size, a dataset of at least 10 times the given size is required before training times of AdaBoost classifier would start to match SVC. Thus, SVC is found to be the best model for the student intervention system context.

5.1 Layman Explanation

The task of a good machine learning classification algorithm is to find the *pattern* between two classes/labels (student passed or not), and given an unknown sample (new student data), be able to reliably and effectively predict the class it belongs to. Support Vector Machine (SVM) based classification algorithms fall under a class of algorithms that tries to predict the *optimal decision boundary* between two or more classes.



(a) Non-SVM approach

(b) SVM approach

Figure 3: An example binary classification scheme with different decision boundaries

The concept of *Optimal decision boundary* could be explained with an example. The two graphs in Figure 3 represent a dataset with two *features* viz., x1 and x2. x1 and x2 for instance could represent "number of past failures" and "number of absences" respectively. Assuming the data points colored *red* in Figure 3 are negative samples and those colored *cyan* are positive samples, intuitively a clear *decision boundary* can be seen between the two sets of data points. A *decision boundary* in this case, is a line dividing the two halves of the graph such that all red points lie on one side and all the cyan lie on the other. When a new sample point is encountered, a *decision* is taken based on the side the new sample point happens to fall into. It is important to realize that there are infintely many ways the graph can be divided up according to the above criteria. That is, any line that is *contained* within the top-left red point and bottom-right cyan point can serve as the decision boundaries. Figure 3a represents three such possible decision boundaries.

Figure 3a can be used to demonstrate the pitfalls of selecting *any* decision boundary that squarely divides up the data points. If the machine learning algorithm chooses Line 1, a new data point that is close to the top-left red point but slightly to its left, could get misclassified as a cyan point because of the decision boundary lying close to the top-left red point. Line 3 also possesses similar characteristics, but differs in that the line lies close to the cyan data points; it might misclassify new samples close to cyan points and the line as red points. Intuitively, thus, line 2 appears to be optimal among the three because it achieves the right *balance* by being the farthest from both the sets of data points. This characteristic is also called *data generalization* and it is achieved by an *optimal decision boundary*. SVM finds the optimal boundary by finding a line between the data points that *maximizes* the *margin* between them. *Margins* can be visualized as the leeway from the decision boundary in which no data point exists. The margins thus will fall on a few data points that are closest to the decision boundary. Figure 3b illustrates this concept. In reality, such *hard margins* are not stipulated in SVM; some tolerance for the formation of margins is accepted depending on the context.

The example in Figure 3 represents a dataset which could be linearly separated (decision boundary is a line). Additionally however, SVM is adaptable to datasets that are non-linearly separable (decision

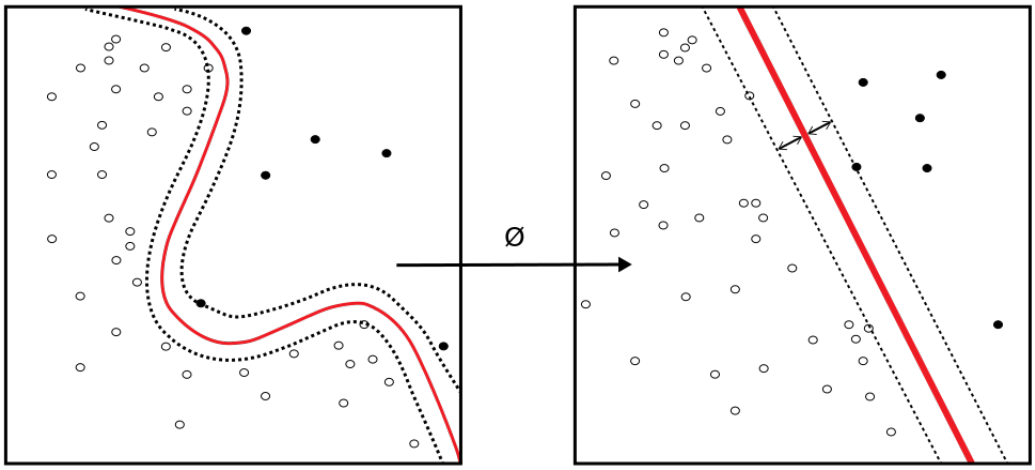


Figure 4: SVM based classifier that employs *kernel trick*. Courtesy: Alisneaky, *WikiMedia Commons*, 2011.

boundary as a curve or some complex shape) too by employing *kernel trick*. Figure 4 illustrates a case where the non-linearly separable data points are *mapped* to a different space using a ϕ kernel function. SVM can be imagined to maximize the margin on the ϕ transformed data points, that have become linearly separable due to the application of kernel. This is an oversimplification because in reality, SVM performs the computations needed for kernel and the optimal decision boundary in a very efficient manner. In effect, SVM can be used for a wide variety of application domains, and investigation (above) with the student data has proved its efficacy for usage in student intervention system.

5.2 Fine-tune and Performance of Best Model

`GridSearchCV()` was used to find the *best* parameters for SVC. Given that the best kernel (RBF) was already found, three parameters viz., `gamma`, `C` and `tol`, were grid-searched. `gamma` influences a sort of *radius of influence* of a data point that is part of the support vectors – lower `gamma` will have smaller influences. `C` parameter defines the smoothness of the decision surface – high `C` generally leading to over-fitting. `tol` is related to the stopping criterion. A certain range around the default values for these parameters were chosen for the parameter grid. The entire training dataset (300 samples) was used, and (standard) scaling on the features was performed before the grid search. The cross-validated grid search yielded an *optimum* classifier with F_1 score of 0.81 with the test set. This indicates that the fine-tuned model performs more or less similar to the default RBF based configuration. Nevertheless, SVC has proven to be the most suitable supervised machine learning algorithm for the student intervention system.

6 Notes

6.1 Single run and 100 runs

To make sure that equivalent model performances are obtained, equally shuffled and split dataset is ensured as input. That is, during single runs (also called sample runs), it is made sure the same dataset split (and shuffle) is performed for all models and in case of 100 runs, the same series of shuffle and split is performed. The seed to the pseudo-random generator that is used for shuffling is adjusted for each run (or block of runs) to maintain this consistency. Additionally, during the split, (reliable) stratification is ensured, and is followed by data scaling where necessary (Ex: SVC).

6.2 General Issues

6.2.1 Degenerate Case

It appeared initially that the dataset given was not really a *good* dataset, in that none of the models seemed to perform very well. Consider the case where all models would predict a positive label regardless of the input. The F_1 score with the entire dataset would thus be $\frac{4}{5} = 0.80$, as the precision would be roughly $\frac{2}{3}$ because data set has $label^+ : label^-$ around $\frac{2}{3}$, and recall would be 1 as it would always predict $label^+$. This case could be called the *degenerate* case, and in some places above, the corresponding F_1 score is referred to as *degenerate score*. Quite annoyingly (initially) none of the models seemed to be performing any better than the *degenerate score*. It was however verified that the models themselves didn't land into degeneracy by only predicting $label^+$, and there were a few false negatives. Ideally, considering that the best model performs just marginally better than the degenerate score, either the need for the complex machine learning could be questioned, or assuming the models are comprehensive, the dataset's representation of required information could be questioned. In either case however, we can't conclude from this that the usage of a prediction system is futile or that dataset is unusable. Confidence, I think, can only be improved with a larger dataset, if available. The *degenerate score* and scaling of a model's F_1 score to match models might not be a standard method – it was thought appropriate as a natural extension.

6.2.2 Dataset balancing

In the case of AdaBoost, the initial experiments with Decision Tree Classifier as the base estimator revealed that it tended to bias towards a few features too often. To counter this, data balancing was performed, by eliminating the dominant samples randomly until their numbers equaled that of non-dominant samples. It was observed that the F_1 scores with the balanced dataset drastically fell down, but it was noted that this was natural both as a consequence of less representation from dominant samples and the total number of *available* training samples being much lower. Similar tests with dataset balancing [Ols05] also indicate decline in accuracy, but not in perceived performance.

6.2.3 Indirect comparison

It is difficult to compare models operating on datasets that vary in their label distribution. Using AdaBoost's performance (in Table 9) as example, a possible mechanism to perform indirect comparison between balanced and unbalanced dataset is derived. An equivalent F_1 score could be calculated by *scaling* the AdaBoost's score with the *degenerate score* for an unbalanced dataset. That is, if the degenerate F_1 score for a classifier operating on a balanced dataset ($label^+ : label^- = 1 : 1$) is $\frac{2}{3}$, the equivalent F_1 score on an imbalanced dataset ($label^+ : label^- = 2 : 3$) will need to be scaled by $\frac{4}{5}$. By this metric, AdaBoost's F_1 score for 200 training points will become 0.74664 from its original 0.62248.

6.3 Neural Network

Use of Neural Networks was considered, but scikit-learn (stable) doesn't directly support multi-layer perceptron currently. Although other libraries (Theano, scikit-neuralnetwork) could be used, exploration has been pushed forward both because it was suggested to "choose 3 supervised learning models that are available in scikit-learn", and neural networks will hopefully be encountered later during *Deep Learning*.

References

[BGV92] Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152. ACM, 1992.

[FS97] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119 – 139, 1997.

[HCL+03] Chih-Wei Hsu, Chih-Chung Chang, Chih-Jen Lin, et al. A practical guide to support vector classification. 2003.

[Joa98] Thorsten Joachims. Text categorization with support vector machines: Learning with many relevant features. *European Conference on Machine Learning (ECML)*, pages 137–142, 1998.

[Joa05] Thorsten Joachims. A support vector method for multivariate performance measures. In *Proceedings of the 22nd international conference on Machine learning*, pages 377–384. ACM, 2005.

[MAP06] Vangelis Metsis, Ion Androutsopoulos, and Georgios Paliouras. Spam filtering with naive bayes – which naive bayes? *Third Conference on Email and Anti-Spam (CEAS)*, 2006.

[Ols05] David L. Olson. *Data Mining and Knowledge Management: Chinese Academy of Sciences Symposium CASDMKM 2004, Beijing, China, July 12-14, 2004. Revised Papers*, chapter Data Set Balancing, pages 71–80. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[Ris01] I. Rish. An empirical study of the naive bayes classifier. *Proceedings of IJCAI-01 workshop on Empirical Methods in AI*, pages 41–46, 2001.

[Sch90] Robert E. Schapire. The strength of weak learnability. *Machine Learning*, 5(2):197–227, 1990.

[Sch13] Robert E. Schapire. *Empirical Inference: Festschrift in Honor of Vladimir N. Vapnik*, chapter Explaining AdaBoost, pages 37–52. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.