UDACITY

MACHINE LEARNING – NANODEGREE

STUDENT INTERVENTION SYSTEM

BY: TODD FARR

1 Machine Learning Problem Type

1.1 Classification vs Regression

This supervised machine learning application is a classification problem. This can clearly be identified by the fact that the target/label is a discrete output. If the goal is solely intervene, then all that's needed to be known is if the student is going to pass or not: "yes" or "no." It however is not required that the model must predict certain percentages or continuous outputs that are associated with Regression problems.

2 Exploring the Data

2.1 Statistical Analysis

The following statistics were calculated from the supplied data set:

- 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 (excluding the label/target column): 30

3 Preparing the Data

3.1 Identify Feature and Target Columns

As identified previously, the total number of features, excluding the label/target column, is 30. Because the main interest was focused on students passing, the "passed" feature was treated as the prediction target for this classification problem. Simple slicing operations were used to separate the original DataFrame into two corresponding *feature* and *target* DataFrames.

3.2 Preprocess Feature Columns

An important step in any machine learning application is to understand the data and how it can be used to accurately predict the appropriate target. Part of this process often includes preprocessing the data into a form that an algorithm can easily decipher and therefore use to draw meaningful conclusions. One common hurdle is categorical features in which the features themselves consist of categorical representations. For simple features (i.e. features with two categories) these can be easily converted into a binary feature. However, for a single feature consisting of multiple categories a common technique is to create as many new features columns as there are categorical choices. These new columns are then individually suffixed with an underscore followed by the name of a single category from this feature. If a data point was classified by one of these categories in the original feature a **1** is

placed in the new corresponding column.   For all other newly created columns this data point is supplied a **0**. A simple tabulated representation of this method can be seen in **Figure 1**.

Figure 1: A tabulated representation of *Dummy Variable* creation & *Binary Data* conversion

3.3 Splitting the Data

SK-Learn's `test_train_split()` method was used to split the dataset into corresponding training and testing sets. This method allows for the input of both test and training sizes which will split the data based on an integer representing the sample size or a decimal value representing a percentage. This method will be particularly useful when testing the various models' ability to learn when subject to smaller training sets while keeping the test set constant.

4 Training and Evaluating Models

4.1 Model Selection

For an appropriate model selection the first step is to identify the type of problem that's needed to be solved. As mentioned in *Section 1.1* of this report, it was already determined that this is a classification problem. For this particular project it was found that the algorithm cheat-sheet form Scikit-Learn was by far the most helpful starting point (Figure 2).
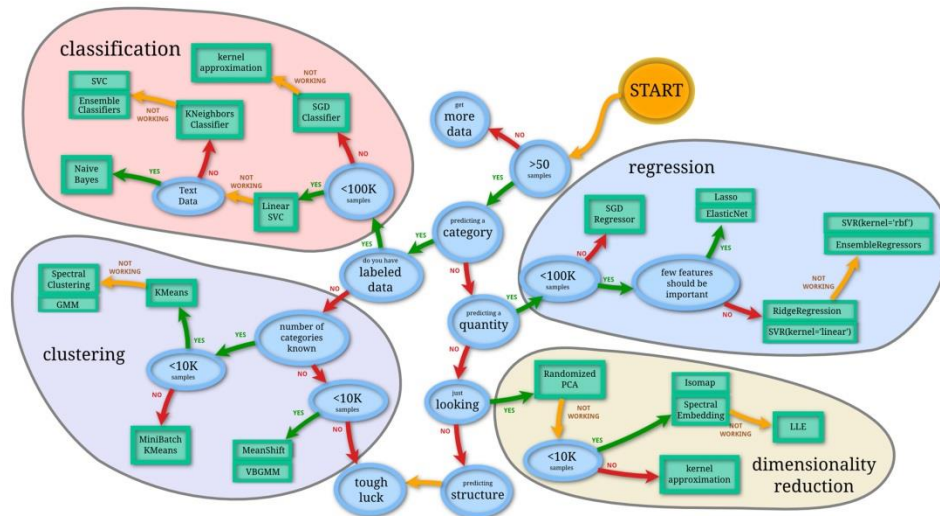
Figure 2: Algorithm cheat-sheet from Scikit-Learn.

Using the knowledge acquired from the course materials and independent research coupled with this map the 3 algorithms selected for this problem were: *Support Vector Machine*, *Naïve Bayes*, and *K-Nearest Neighbors*.

1. ***Support Vector Machines (SVMs)*** are supervised learning algorithms that map training inputs to representations of two (or more) categories as points in space. These mappings attempt to maximize the distance between the points representing the two categories so that a clear boundary is defined. The width of this boundary is often referred to as the margin.

    a. *Strengths:* SVMs work very well in complicated domains where there are clear margins of separation between classifications. Also, since their sole function is to maximize these margins they create decision boundaries that have the tendency to be highly accurate at classifying labels for new inputs.

    b. *Weaknesses:* Because feature mappings to higher spaces can be complex if the base space is not linearly separable, SVMs can be difficult to tune and hard to interpret. Also, with this added complexity they have the tendency to have high memory taxation and run the risk of overfitting the data.

    c. *The reason behind the choice:* The most efficient way to train a SVM is to use linear decision boundaries which can be done using the **SVC** class in SK-Learn and assigning it's kernel to *'linear'*. However, because this class functions using the *libsvm* library, which also supports non-linear kernels, there can be hidden computational costs. Therefore, the **LinearSVC** class, which functions on the more efficient *liblinear* library, was chosen in-line with the cheat-sheet mapping from above. In addition because SVMs can be tuned via the kernel to better fit the data, this algorithm can serve as a good backup plan if the other two selections don't perform as expected.

2. ***Naïve Bayes*** is a supervised learning algorithm built on the foundation of the Bayes Theorem with the "naïve" bit referring to the fact that it assumes the independence between each pair of features.

a. *Strengths:* For Naïve Bayes its strength is simplicity, which in turn means it can be extremely fast when compared to more complex algorithms. In addition it generally requires a smaller amount of data to estimate parameters. Also, because each feature distribution can be evaluated independently it's less prone to suffer from the curse of dimensionality.

b. *Weaknesses:* Naïve Bayes Classifiers weakness actually lies in its strength; because of its simplicity it cannot learn the relationship between features. This can be ultimately attributed to its assumption of feature independence.

c. *The reason behind the choice:* This choice is actually a deviation from the SK-Learn cheat-sheet map as all text data has been preprocessed to binary and dummy variables. Its selection relies solely on the fact that it's simple and quick. However, despite the over-simplified assumptions and resulting theoretical disadvantages, Naïve Bayes has been proven to work well in real-world applications. This coupled with its ability to converge quickly on a classifier with less data makes it seem like it's a good candidate for this application where computation time needs to be closely monitored and the number of students in any given district is relatively limited.

3. **K-Nearest Neighbors** is an instance-based or non-generalizing supervised classification algorithm. This means that it relies on simply "remembering" the training data rather than fitting a function or generalization to it. KNN then uses this remembered data to label new inputs by querying the labels of its nearest "neighbors." The neighbors are those data points that have the most similar features (the closest in distance) to that of the input.

a. *Strengths:* For K-Nearest Neighbors it's simple, easy to understand, it's effective if the training dataset is large, and because it's only "remembering" it's computationally less expensive to train than those algorithms that need to generalize the data. It can also be fairly robust if the training data is noisy.

b. *Weaknesses:* K-Nearest Neighbors is referred to a lazy-learner because it actually doesn't do any learning until it's time to predict labels for new inputs. Therefore, this can be computationally expensive at prediction time because in theory all distances from the training data points to the input need to be calculated at the time of the query. There are advanced ways of indexing the data to help alleviate this problem, but this point should still be considered. Additionally KNN assumes that those data points that are close in proximity are similar. Lastly, it's important to note that the k value can have large effects on this algorithms performance and it can be difficult to tune this parameter.

c. *The reason behind the choice:* In addition to the continuation of following the recommendations from the algorithm map, the K-Nearest Neighbors algorithm is easy to teach, it's assumed that students at risk share have the same key features (data points near each other are similar) and since this particular dataset is small the power and time needed for queries will be limited.

4.2 Fitting the Models

The three previously mentioned models were fit to the data using the SK-Learn library classes with default settings. To study the effects that training size has on each model's performance and the associated training time, the model was taught three separate times using training sizes of 100, 200 and 300 examples. For reference, the test set remained fixed at 95 examples. The functions were written as an iterative process so that the data was re-shuffled and re-split when a new training size was supplied. Each model's associated values can be seen in the tables below (Tables 1-3).

Linear SVC	Training Set Sizes		
	100	200	300
Training time (secs)	0.026	0.069	0.117
Prediction time (secs)	0.001	0.002	0.013
F1 score for training set	0.932	0.808	0.833
F1 score for test set	0.723	0.740	0.750

Table 1: The data associated with the SVM LinearSVC model

Naïve Bayes	Training Set Sizes		
	100	200	300
Training time (secs)	0.003	0.004	0.003
Prediction time (secs)	0.001	0.001	0.002
F1 score for training set	0.354	0.788	0.816
F1 score for test set	0.365	0.794	0.718

Table 2: The data associated with the Gaussian Naive Bayes model

K-Nearest Neighbors	Training Set Sizes		
	100	200	300
Training time (secs)	0.002	0.002	0.004
Prediction time (secs)	0.007	0.012	0.031
F1 score for training set	0.847	0.870	0.870
F1 score for test set	0.718	0.778	0.780

Table 3: The data associated with the K-Nearest Neighbors model

The F1 score is the weighted average of the precision and recall of the associated model in question. The highest value that the F1 score can assume, meaning perfect recall and precision, would be 1. On the other hand the lowest value (associated with poor model performance) which can be represented is 0. The formula for the F1 score can be seen in Equation 1.

$$F1 = 2 * \frac{preicsion * recall}{precision + recall}$$ (Eq. 1)

Precision and recall in this instance can be defined by Equations 2 & 3, where t_p is the number of True Positives (the classification label matches the predicted label), f_p is the number of False Positives (the

classification is predicted to be *True*, but is actually *False*), and f_n is the total number of False Negatives (the classification is predicted to be *False*, but are actually *True*)

$$precsion = \frac{t_p}{t_p + f_p}$$ (Eq. 2)

$$recall = \frac{t_p}{t_p + f_n}$$ (Eq. 3)

5 Choosing the Best Model

5.1 Final Model Selection

The highest of all computed F1 scores with default parameters belongs to the Naïve Bayes algorithm at a training size of 200 examples with a value of 0.794. In this particular example, when benchmarked against the other two models, training time for Naïve Bayes wasn't the fastest but it was equivalent to the next highest F1 score of 0.780 belonging to KNN.

As expected, because of its simplicity, the most efficient model tested was the Gaussian Naïve Bayes classifier. It secured both the fastest combined training and query times with positive prediction results on the test set of 0.794 and 0.718 for the 200 and 300 example-sized training sets, respectively. It however could not generalize the data given the smaller set of 100 example in which it return the lowest score recorded of 0.365.

K-Nearest Neighbors was penalized on queries, as hypothesized previously, with a 0.018 second increase (138%) over the next query intense model on the training size of 300. Despite the increase, it did perform well on this training set size with an associated F1 score 0.780. Since KNN results can be dependent on the selection of an appropriate K-value this model could probably be tuned to achieve better results.

Considering all this information it was decided to proceed with using Support Vector Classifiers. While the added computational cost seems to steer away from the requirements, it's difficult to put a price on the success of the students and meet the goal of a 95% graduation rate within the decade if indeed the most accurate model requires these additional resources. While Naïve Bayes seems like another realistic choice its lack of tunable parameters limits its ability to be improved from its out-of-the-box performance. KNN may seem like another logical choice; however we may miss the students that don't necessarily exhibit the classic warning signs.

5.2 What is a SVC?

A Support Vector Classifier is a machine learning classification algorithm that attempts to draw linear decision boundaries that maximizes the margins between the classifications. This process of maximizing the margin can best be understood in conjunction with the accompanying visualization (Figure 3).

In this figure, two separate classifications can be represented by either a Blue Circle or Red Ex. These two classification are clearly linearly separable (a straight line can be drawn to divide them). However, because this line can take of the form of any infinite number of lines it's important to choose the best

one. In Fig. 3 two of these lines are depicted. On the left side the decision boundary does in fact separate the two classifications but comes within close relative proximity to both a circle and an ex. Now imagine if an algorithm settled on this line and it then needed to classify a new point represented by the white square. It's already known that it is supposed to be a blue circle, but because the algorithm did not choose the line that maximizes the margin it would classify it incorrectly as a Red Ex. Luckily because Support Vector Machines do draw the line that maximizes the margin (the depiction on the right) if we add that same new data point in the same location it's now correctly classified as a Blue Circle.
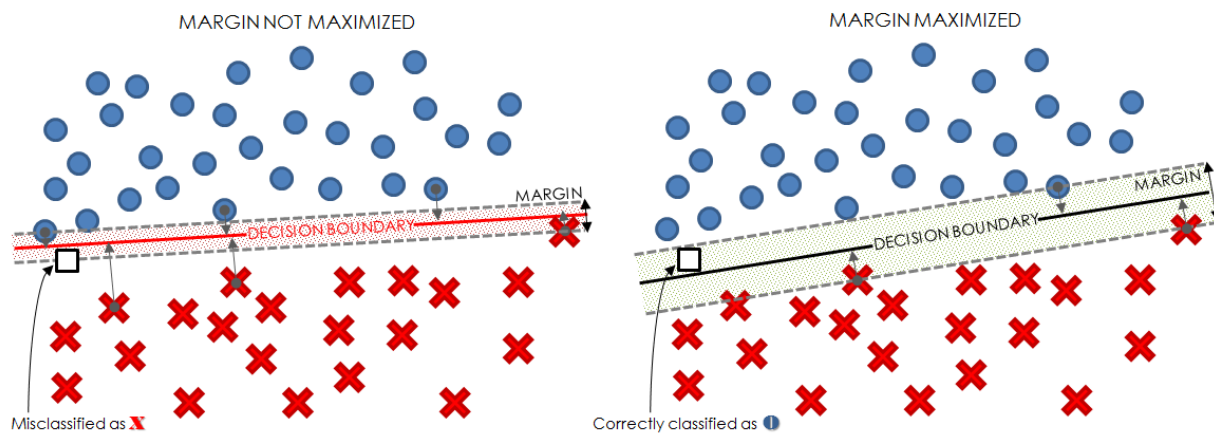

Figure 3: SVM margin maximization visualized.

This can be an extremely effective method for data with clear decision boundaries, but what if the data is not linearly separable? This is where SVMs become really powerful and offer high tunability. Using a method referred to as 'kernel-trick,' SVMs can actually map data points to new spaces of higher dimensions. This is best visualized by an example mapping data from a 2-dimensional space to one of 3-dimensions as seen in Figure 4. Because the points have been mapped to a higher dimension the "linear" separator is also elevated to that of a hyperplane. Of course SVMs are capable of mapping data points to higher (even infinite) dimensions but it's increasingly difficult to imagine or visually represent.
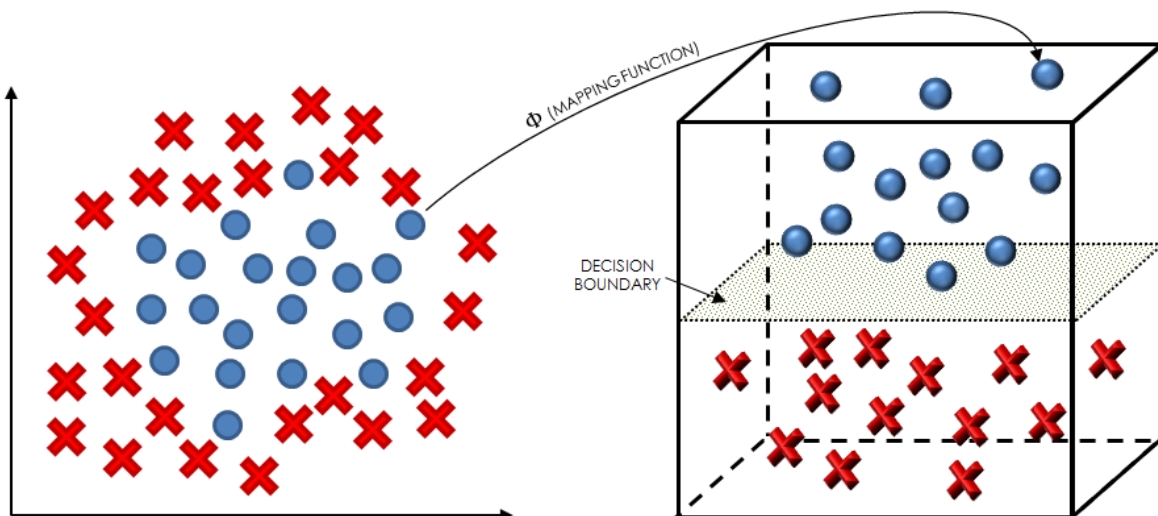

Figure 4: The "kernel-trick" visualized

5.3 Tuning the SVC & the Final Model

The final model was tuned using GridSearchCV, which accepts a model and iterates over a supplied dictionary of tunable parameters in attempt to find the optimal ones that returns the best cross-validation estimator performance score from the supplied metric (F1 Score in this case) on the given dataset. The parameters provided where different kernels ('linear', 'poly' and 'sigmoid') in an attempt to leverage the aforementioned kernel trick. The results and final model can be seen below:

```
***** FINAL MODEL *****
GridSearchCV(cv=None, error_score='raise',
        estimator=SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0, degree=3, gamma=0.0,
   kernel='rbf', max_iter=-1, probability=False, random_state=None,
   shrinking=True, tol=0.001, verbose=False),
        fit_params={}, iid=True, loss_func=None, n_jobs=1,
        param_grid={'kernel': ['rbf', 'poly', 'sigmoid']},
        pre_dispatch='2*n_jobs', refit=True, score_func=None,
        scoring=make_scorer(f1_score), verbose=0)

Best Kernel: {'kernel': 'sigmoid'}
Best Score: 0.8030296639625579
```

The kernel that achieved the best F1 Score output from GridSearchCV of 0.803 was the 'sigmoid' kernel. This is a 7% increase in accuracy over the LinearSVC which was evaluated before tuning. It's also the highest score out of any of the previous classifiers tested on the dataset. A new model was created using SVC with the kernel set to 'sigmoid' and then ran through the same scenarios as above. The final results can be seen in Table 4.

SVC kernel = 'sigmoid'	Training Set Sizes		
	100	200	300
Training time (secs)	0.002	0.007	0.014
Prediction time (secs)	0.002	0.009	0.004
F1 score for training set	0.857	0.795	0.793
F1 score for test set	0.750	0.841	0.834

Table 4: The final SVC with kernel set to 'sigmoid'