KOBE BRYANT SHOT SELECTION

Authors: Pedro Marcelino and João Coelho

Email: pmarcelino@gmail.com and joao.pedro.cp.coelho@gmail.com

Personal websites: www.pmarcelino.com and www.joaopcoelho.com

Date: 01/07/2016

AUTHORS INFORMATION

Pedro Marcelino is a PhD student at IST, Lisbon (Portugal). He applies computational methods in the field of transportation engineering. He has a yearlong experience in Python programming and he has followed a self-learning path to develop practical applications of machine learning and data science. Talking and writing about machine learning, data science and Python really motivated Pedro to start the *HAL Project* blog, where he reports his steps in his 'heuristic approach to become a Machine Learning expert'.
He is also active in the online community, contributing to Stack Overflow, Quora and open source projects, as well as participating in machine learning competitions, such as Kaggle. In his free time, he likes to play sports, study French and do public speeches.

João Coelho is a research fellow at LNEC, Lisbon (Portugal). His research focuses on high performance computing approaches to simulations of real-life engineering systems, such as the behaviour of large concrete dams over time or the optimization of energy-efficient building design. He completed a MSc in Physics and has since developed an interest in data science and machine learning. In his free he enjoys playing basketball, travelling and learning about other people, places and cultures.

ABSTRACT

This paper aimed to report our participation in a Kaggle competition named ‘Kobe Bryant Shot Selection’. The objective of this competition was to predict Kobe Bryant’s probability of making a field goal using machine learning techniques.

By employing a ‘learn by doing’ approach, we developed a solution to the problem. Two models were applied in this project. One was based on the k-Nearest Neighbors (k-NN) algorithm and another on the Support Vector Classification (SVC) algorithm.

We conclude that the SVC algorithm was able to perform better predictions. According to this model, we are able to predict the outcome of Kobe’s shots with 60.7% accuracy. This model allowed us to achieve the 285th position in a total of 1117 participants in the Kaggle competition.

This study was a first step in our way to become machine learning experts. In that sense, we also conclude about the importance of the preprocessing step and computational efficiency. This conclusions will be useful to guide our future work.

INTRODUCTION

The purpose of this paper is to report our participation in a Kaggle competition. The chosen competition was the ‘Kobe Bryant Shot Selection’, which intended to predict a probability that Kobe Bryant made the field goal.

By employing a ‘learn by doing’ approach, we attempted to solve the problem and expand our current machine learning knowledge. Accordingly, this project was an exercise that allowed us to, in a practical way, understand the machine learning process, learn some concepts and guide our future study.

This paper begins by a description of the Kaggle competition we enrolled in. It will then go on to a detailed presentation of the methods used to solve the problem, specifying what we did in each part of this project. After, we present the results obtained and, finally, we take the conclusions and report our future work intentions.

BACKGROUND

On Wednesday, April 12, 2016, Kobe Bryant retired from the NBA. In his last game as Los Angeles Lakers player he scored 60 points. He was drafted into NBA at the age of 17 and, throughout his 20-year career, Kobe earned the sport's highest accolades.

The specific objective of the Kaggle competition 'Kobe Bryant Shot Selection' was to predict a probability that Kobe made the field goal. We became interested in this competition because it is well suited for practicing classification basics, feature engineering, and time series analysis. Moreover, as scientists and former basketball players, we are always glad to apply scientific knowledge to our favourite sport.

For this competition, a dataset containing the location and circumstances of every field goal attempted by Kobe Bryant was available, along with an indication of whether that field goal was made or missed (the label shot_made_flag). From a total of 30697 career shots, a total of 5000 had their shot_made_flag labels removed, in order to create the test set shots for which we must submit a prediction. The dataset included the following fields: action_type, combined_shot_type, game_event_id, game_id, lat, loc_x, loc_y, lon, minutes_remaining, period, playoffs, season, seconds_remaining, shot_distance, **shot_made_flag**, shot_type, shot_zone_area, shot_zone_basic, shot_zone_range, team_id, team_name, game_date, matchup, opponent, shot_id.

This competition started at 12:50 pm, Friday 15 April 2016 and ended at 11:59 pm, Monday 13 June 2016. Kaggle users are allocated points for their performance in competitions and are separated into three tiers: Novice, Kaggler and Master. This competition did not award ranking points and did not count towards tiers.

METHODS

This project followed the process suggested by Raschka (2014) for the development of machine learning systems in predictive modeling. Figure 1 presents a typical workflow diagram of the suggested process.

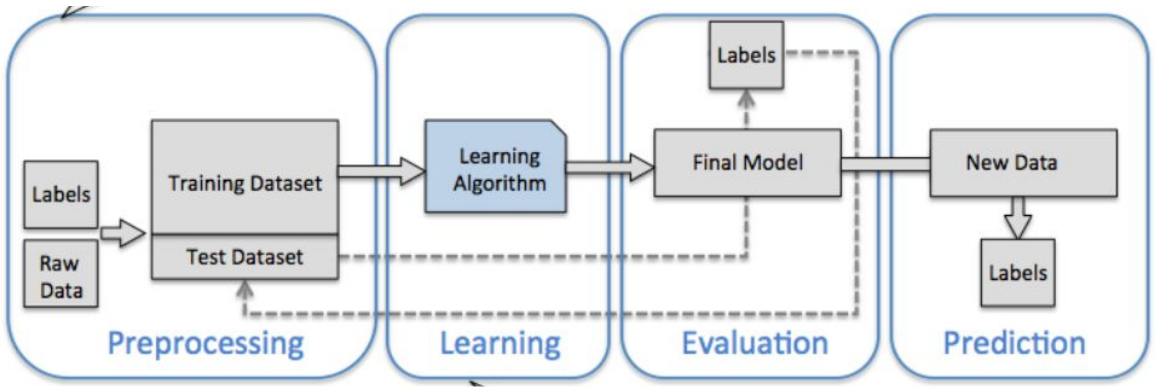

Figure 1 - Roadmap for building the machine learning system (Raschka 2014).

As we can see, the first step in this process is data preprocessing, which includes feature selection, feature extraction and scaling. Afterwards, a learning algorithm is applied and a final model is defined using the training dataset. Once we have a final model, we are able to evaluate it using the test dataset. When satisfied with the final model, we are able to use it and make predictions on new data.

In the context of this work, the training dataset consists of 25697 data points, corresponding to the same amount of Kobe shots, each labelled has having been successful or not. Since this dataset is not very large, we use cross-validation on the training set to simulate the validation set whenever models have to be evaluated before their final form (for example, when deciding on model parameters), rather than strictly partitioning the training set into two distinct sets.

The new data on which we'll make predictions is the test set, which encompasses 5000 data points without their respective labels; in order to evaluate our model's performance on this set, which effectively amounts to its final evaluation, we must submit our predictions to the Kaggle platform.

The following sections detail some aspects related with preprocessing, learning, evaluation and prediction.

Preprocessing

A primary concern of any machine learning project is data quality. The learning ability of an algorithm depends on it as well as on the amount of useful information it contains. The problem is that data rarely comes in the form and shape that is necesssary. Data is often incomplete, redundant and inconsistent. Accordingly, it is necessary to analyse and preprocess a dataset before we use it on a learning algorithm (Christobel and Sivaprakasam 2012).

Common steps in data preprocessing include tasks to clean data, integrate data, transform data, reduce data and discretize data. These steps lead to a final dataset that can be used for algorithm learning purposes.

In our project, the following preprocessing tasks were performed:

- Handling missing data;
- Feature reduction (selection and extraction);
- Handling categorical features;
- Standardization.

Handling missing data

Processing missing data is the most important task in data cleaning (Christobel and Sivaprakasam 2012). Since the dataset used in this project was priorly processed by Kaggle, missing data is not a significant issue. The existing missing data refers to the data used by Kaggle to evaluate the models developed by the users. Accordingly, we deleted from the dataset all the entries with missing values.

Feature reduction

In machine learning, a feature is the specification of an attribute and its value (Kohavi and Provost 1998). It can be said that the concept of 'feature' is to machine learning algorithms as the concept of 'independent variable' is to statistical techniques.

By definition, feature reduction aims to reduce the number of features used in model construction. Feature reduction is useful because it simplifies models, making them easier to interpret; shortens training times; and enhances generalization by reducing overfitting.

Feature reduction can result from feature selection and/or feature extraction. Feature selection is selecting a subset of existing features without a transformation. Feature extraction is building a new set of features from the original feature set.

The process followed in this project for feature selection was based on experimentation, mainly guided by our experience as basketball players and coaches. Accordingly, we defined a set of hypothesis (e.g. if the player is closer to the basket, the probability of making the field goal is higher), we plotted the data and we analysed the charts to see if there was any graphical evidence of our hypothesis. We also standardized the dataset and used the Pearson correlation coefficient to rank variables, filtering those with a low score.

Figures 2 and 3 show two examples of the analysis undertaken in order to identify relevant features. In Figure 2, Kobe's shot percentage is plotted against shot distance, and a clear relationship is readily identified - the farther away from the basket Kobe shoots, the lower his shot percentage is. Thus shot distance is a useful feature to consider in any model.

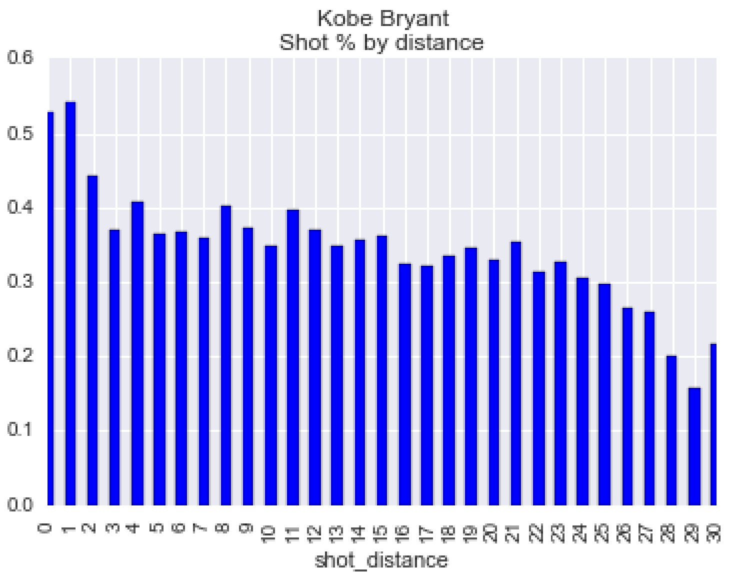

Figure 2. Kobe’s shot percentage by distance.

In stark contrast, Figure 3 shows that Kobe’s percentages are unchanged as a result of the season stage, *i.e.*, there is no meaningful difference between his performance in regular season versus playoff games. Hence, whether the game is a playoff game or not does not seem to be a relevant feature to consider in our models, which is somewhat surprising given our intuition that Kobe’s performance improves on the grand stage.

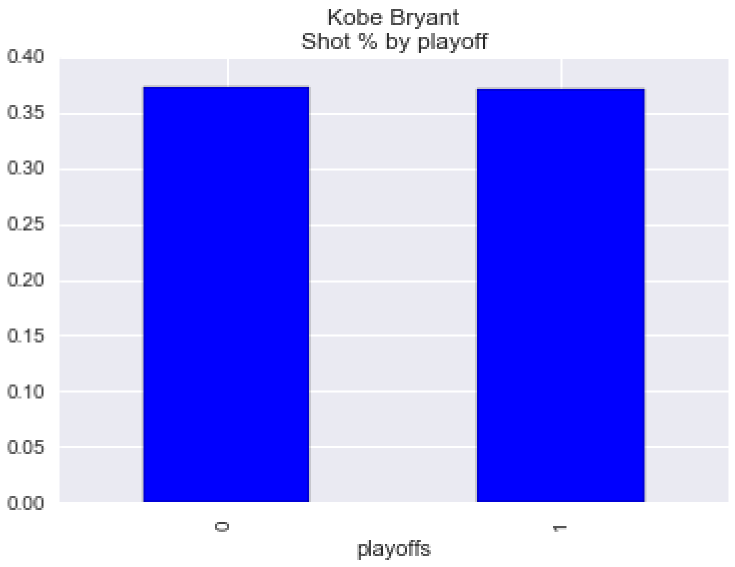

Figure 3. Kobe’s shot percentage by type of game (playoff/non-playoff).

Regarding feature extraction, we created the ‘angle’ feature. This new feature resulted from the combination of ‘loc_x’ and ‘loc_y’. The rationale behind the creation of this feature was that from certain angles it is easier to score, which was intuition we had from our experience. Further analysis of the Pearson correlation coefficient supported this assumption.

After feature selection and extraction, we selected the following set of features to address our problem:

- Shot distance;
- Shot angle;
- Game time remaining;
- Game period;
- Game location (home/away);
- Season.

Handling categorical features

Data transformation was applied to convert some categorical features into numerical, as numerical features are more appropriate for certain algorithms. For example, the ‘matchup’ feature was converted to associate value 0 to ‘home’ and 1 to ‘away’.

Standardization

Kaggle dataset had features with different scales. This poses a problem because most machine learning algorithms perform better if features are on the same scale (decision trees and random forests being some of the very few exceptions) (Raschka 2014). For example, an algorithm that uses a distance function in its optimization is biased if one feature varies from 0 to 1 and other varies from 0 to 100 000, as the feature with the larger magnitude can dominate the function and make the estimator unable to learn from other features as expected.

There are numerous techniques to avoid this problem. In this project we used the standardization technique. Standardization is a data transformation procedure that centers - by removing the mean value of each feature - and scales the data - by dividing non-constant features by their standard deviation - so that feature columns take the form of a standard normal distribution.

One of the reasons why we used standardization was its performance with linear models (e.g. Support Vector Machines (SVM)). In linear models, weights are initialized to 0 or small random values, which makes it easier to learn the weights if features columns has a standard normally distributed form. Moreover, standardization has useful information about outliers and makes the algorithms less sensitive to it (Raschka 2014).

Learning

A learning algorithm defines a mathematical model through learning from some sort of observations or data. These models can be used for different purposes, such as classification, regression, or clustering.

There are two main types of learning: supervised learning and unsupervised learning. Supervised learning refers to those cases in which we have a set of outcomes that can guide us in model building (e.g. classification, regression). On the contrary, unsupervised learning refers to those case in which we do not have outcomes and our goal is to identify data structures from the existing features (e.g. clustering).

In this project, we used two different learning algorithms: k-Nearest Neighbors (k-NN) and Support Vector Classification (SVC). The choice of the algorithms was based on the scikit-learn flowchart (scikit 2016). Algorithms are described in the following sections.

k-Nearest Neighbors (k-NN)

The k-Nearest Neighbors (k-NN) algorithm classifies instances according to the classification of the closest instances in the problem space. It is based on the principle that, in a dataset, instances will be close to other instances (nearest neighbors) with similar properties (Kotsiantis 2007)

Since this project intended to compute probabilities, k-NN was used for regression. This means that the probabilistic label value of an unclassified instance was determined by the average value of the its *k* nearest neighbors label, where *k* is always a positive integer. Detailing, the k-NN algorithm is as follows:

1. Use cross-validation to define the best *k* (number of nearest neighbors);
2. Use a distance measure (e.g. Euclidean) and calculate the distance between the query instance and all the training samples;
3. Sort by distance and choose the *k* training samples with minimum distance;
4. Define label probabilistic value according to the average values of the nearest neighbors categories.

Figure 4 illustrates the principle of the algorithm.

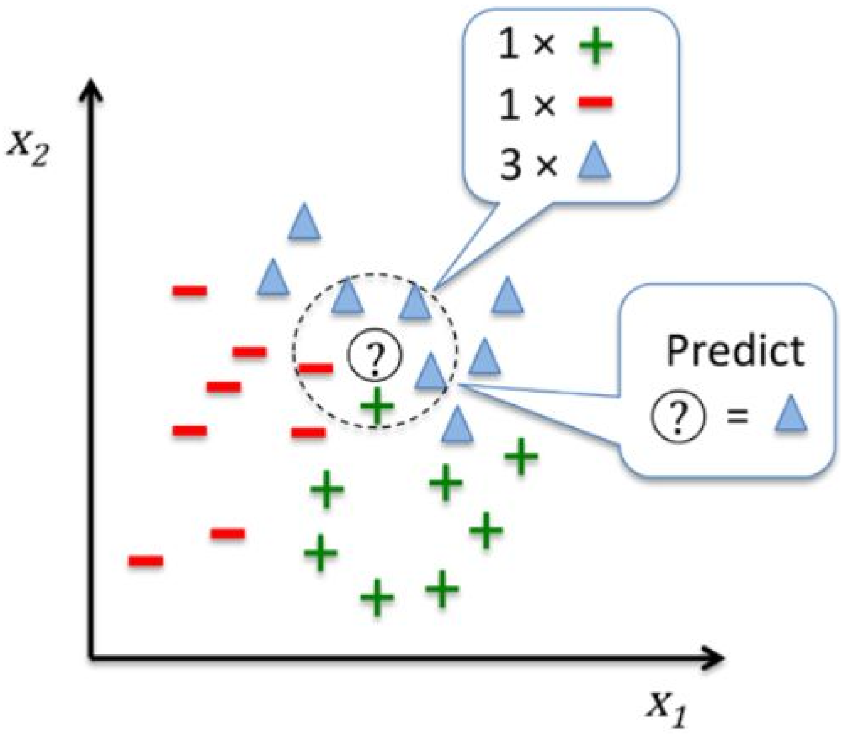

Figure 4. k-NN principle (Raschka 2014).

Support Vector Classification (SVC)

The Support Vector Classification (SVC) algorithm is based on the construction of a hyper-plane or set of hyper-planes that operate like borders: entity classification will depend on the side of the border the entity is.

This algorithm is an implementation of the Support Vector Machines (SVM) algorithm. Figure 5 exemplifies the algorithm application for a two-class problem. The same principles are applied for problems with more classes.

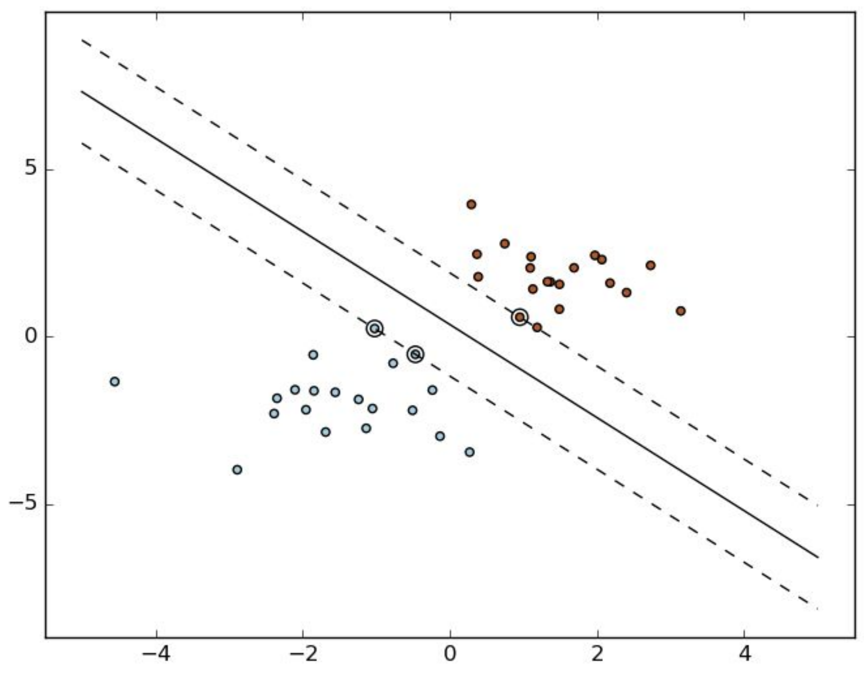

Figure 5. Example of SVC application.

The application procedure followed was based on the recommendations given by Hsu *et al.* (2016):

1. Transform data to the format of an SVM package (we used the libraries that come with scikit-learn);
2. Conduct simple scaling on the data (as described in Preprocessing);
3. Consider the RBF kernel K(x; y) = $e^{-\gamma\|x-y\|^2}$;
4. Use cross-validation to find the best parameter C and γ (it was not possible because of computation power was lacking);
5. Use the best parameter C and γ to train the whole training set (we used the default parameters).

Evaluation

Model evaluation is about estimating model performance on new data. To fit a model, a training data set is used. The model is then applied to a test set to estimate how well it performs on unseen data. If the model has a satisfactory performance, we can use it to make predictions.

According to Kaggle rules, models were evaluated on the log loss. Log loss is the logarithm of the likelihood function for a Bernouli random distribution. This error metric is common when we need to predict that is something is true or false with a probability (likelihood) ranging from definitely true (1), to equally true (0.5), to definitely false (0). Log loss is given by Equation 1:

$$logloss = -\frac{1}{N}\sum_{i=1}^{N}(y_i \log(p_i) + (1-y_i)\log(1-p_i)) \quad (1)$$

The use of log on the error provides extreme punishments. In the worst possible case, a single prediction that something is definitely true (1) when it is actually false will add infinite to your error score and make every other entry pointless. To avoid such punishments, Kaggle competitions predictions are bounded away from the extreme values by a small value.

It is important to remark that, when evaluating a model, the goal is to minimize the resulting generalization error and not the error on the training data (Kearns et al. 1997). Accordingly, a k-fold cross-validation was used in this project to optimize the selection of model parameters.

A k-fold cross-validation help us to obtain reliable estimates of the model's generalization error. To perform k-fold cross-validation the training data set is splitted into *k* folds and 1-*k* folds are used to train the model, while the remaining 1 fold is used to test the model. This procedure is repeated *k* times and the average performance of the models is calculated. The concept behind k-fold cross-validation is summarized in Figure 6 for a *k* = 10 case:

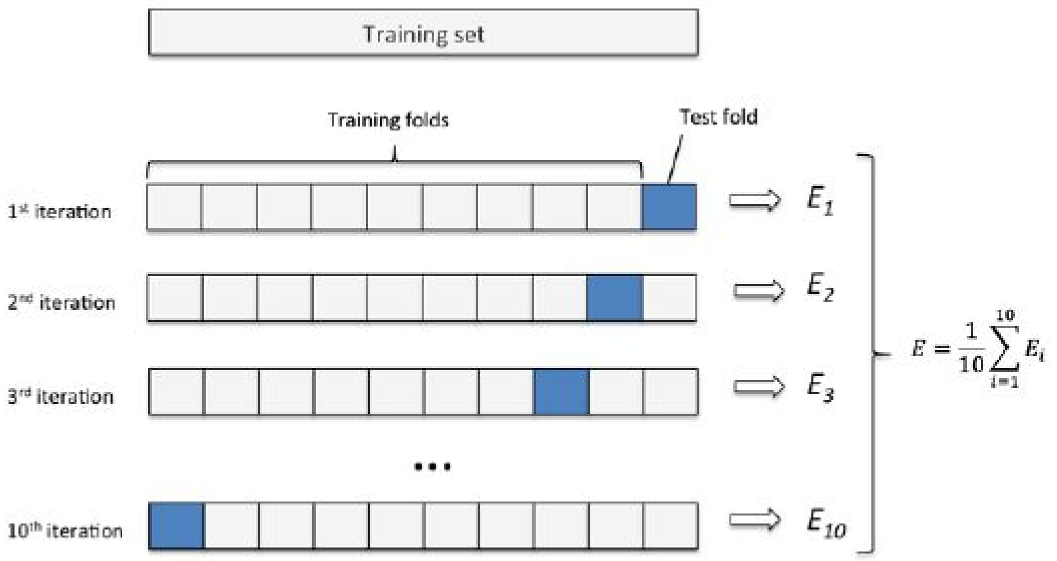

Figure 6. Concept behind k-fold cross-validation (Raschka 2014).

As recommended by Raschka (2014) for problems with large datasetes, we opted for 5 folds in our project. This *k* value allows for an accurate estimate of the model, while reducing the computational cost of refitting and evaluating the model on the different folds.

RESULTS AND DISCUSSION

We trained several models which relied on one of two algorithms: k-Nearest Neighbors (k-NN) and Support Vector Classification (SVC).

k-Nearest Neighbors

We used the scikit-learn implementation of k-NN to train our classifier. Before training our model, we applied feature normalization to our training set via the scikit-learn method StandardScaler, which standardized our features.

The first task was to choose the number of neighbors *k* to consider. Although model performance generally increases with the number of neighbors, so does the computational cost of training the model. Thus an optimum value of *k* that balances these two factors must be found.

To address this problem we trained a different model we used 5-fold cross-validation to define a cross-validation set and then trained a model for every value of *k* from 1 to 30. Each model's performance was evaluated by computing the logloss for each of the cross-validation folds, and a mean logloss was computing by averaging over all folds. Figure 7 shows the mean logloss obtained for each *k*.

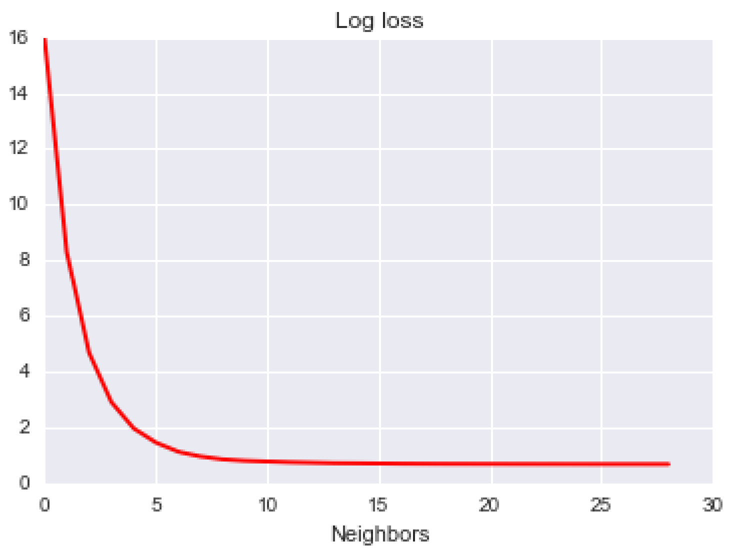

Figure 7. Log loss according to the number of neighbors .

These results suggest that the optimum *k* is somewhere around 10, as for greater values performance does not seem to improve. We therefore chose k=10 as the number of neighbors to consider in our model.

Upon evaluation on the test set (through a submission to the Kaggle competition), this model yielded a logloss of 0.79892, which is a poor result considering that random guessing results in a logloss of 0.69315.

Since this was not a satisfying result, we decided to overthrow our choice of best model and experiment models with different values of *k*, evaluating them on the test set (through submission to the Kaggle competition). Although this is not the best way to select our model, since it might overfit to the test set, it was a fruitful way of understanding the impact of *k* in our algorithm's performance.

Table 1 summarizes the results obtained for the different models.

Neighbors	**Logloss**
4	2.83585
8	0.90632
10 (initial choice)	0.79892
16	0.71033
18	0.70840
50% Benchmark	0.69315
28	0.68999
43	0.67862
50	0.67958
100	0.67799
500	0.67872
1000	0.68149

Table 1. Log loss values for k-NN models.

Two conclusions can be drawn from this analysis. First of all, it shows that our decision on the optimum value for *k* was slightly innacurate, since it yielded a logloss with significant room for improvement. However, it supports our rationale for that decision, since model accuracy improved at most by 0.12 for *k* greater than 10, an improvement of 17.7%; in contrast, lower values of *k* yielded much worse model performance.

In order to choose a better value of *k*, we should have inspected the results depicted in Figure 7 in more detail, which would have enabled us to find the actual value of *k* above which model performance becomes stagnant.

Secondly, it shows that this algorithm cannot perform much better than the 50% benchmark regardless of how its single parameter k is tuned, and is therefore an inadequate choice for this problem.

The two main ways in which this model could be improved would be to improve its feature selection, as described in the Methods section, or to use a classification algorithm that better suits this problem. We decided to keep our features unchanged and try a new algorithm.

Support Vector Classification

As an alternative to k-NN, we decided to use a SVC method with RBF kernel using the scikit-learn implementation. SVC with RBF kernel has two free parameters: C, the penalty parameter of the error term, and gamma, the coefficient for the RBF kernel.

On a first try we assumed the default values for both C and gamma and trained a classifier on the whole training set. This yielded a logloss value of 0.66703, better than any of the values obtained for the k-NN method.

In order to obtain better parameter values for C and gamma, we attempted a grid search on both C and gamma. However, because of a lack of computing power, we could not complete this optimization step.

As a result, we were able to conclude that SVC was a better algorithm than k-NN for this classification job and the best logloss value we were able to obtain was 0.66703. We were unable to optimize the parameters for SVC but remain optimistic that such a process would yet more adequate values of C and gamma and thus a better model for our problem.

Misclassification error rates

In addition to the logloss, another convenient way of evaluating our model is to analyse its misclassification error, i.e., the fraction of test samples whose label is correctly predicted by our model. Since we have no direct access to the test set (we can only access it via Kaggle), we performed cross-validation on the training set to obtain the misclassification error for the best model of each algorithm.

In the case of k-NN, the best model was obtained for 100 neighbours and this yielded a misclassification error of 41.1%. In the case of SVC, the best model was obtained for the default parameters and yielded a misclassification error of 39.3%. This means that through our best model, SVC, we are able to accurately predict the outcome of 60.7% of Kobe’s shots.

CONCLUSIONS AND FUTURE WORK

This study set out to determine the probability that Kobe Bryant made the field goal using two models: the k-NN and the SVC model. The results indicate that the SVC model developed has a better performance than the best k-NN. According to this model, we are able to successfully predict the outcome of of Kobe's shots.

Future work should focus on determining how to improve the preprocessing step. We feel that about 90% of our work was related to this step and almost everything we did followed an intuitive process. Since preprocessing is essential in any machine learning problem, we intend to learn more about it.

It is also important to note that the SVC model we developed was computationally heavy. Every run took hours to be performed and we even had to cut the grid search optimization we had because it was just too heavy. We believe the main reason behind this situation was the repetition of the normalization step in every run. This situation stated the importance of algorithm optimization.

Finally, we also might explore time series in the near future. Although it was one of the objectives of this competition, we did not have time to develop a solution that used time series analysis. We are aware of how this analysis could improve our results, so we will study it hereafter.

In conclusion, we must say that regarding the Kaggle competition, our solution allowed us to achieve the 285th position in a total of 1117 participants. However, the biggest result we take from this project was the knowledge we acquired during the process. It made it worthwhile.

REFERENCES

Raschka, S. (2014). Python Machine Learning. PACKT Publishing.

Christobel, Y., & Sivaprakasam, P. (2012). Improving the performance of k-nearest neighbor algorithm for the classification of diabetes dataset with missing values. International Journal of Computer Engineering & Technology, 3(3), 155–167.

Kohavi, R. & Provost, F. (1998). Glossary of terms. Machine Learning, 30, 271-274.

Kotsiantis, S. B. (2007). Supervised machine learning: A review of classification techniques. Informatica, 31, 249–268.

Ben-Hur, A., Horn, D., Siegelmann, H., & Vapnik, V. (2001). Support vector clustering. Journal of Machine Learning Research, 2, 125–137.

scikit-learn (scikit), Flow Chart - Choosing the right estimator. Available from: http://scikit-learn.org/stable/tutorial/machine_learning_map/index.html [Accessed 10 Jun 2016]

Kearns, M., Hill, M., Mansour, Y., Aviv, T., Ng, A. Y., & Ron, D. (1997). An Experimental and Theoretical Comparison of Model Selection Methods. Machine Learning, 27, 7–50.