Machine Learning Methods for Multi-League Football Match Result Prediction

**The problem:**

While there are several papers on football match result prediction, many (if not most) usually have limited scope, like say focusing on a single football team, season, competition, etc. Because of this, you cannot use these systems for prediction outside the scope they were trained for or at least you would have to train individual models for say 10 teams if you want to support 10 teams.

**Aim and objective of the paper:**

The aim of the paper is to develop a generalized model that can make match result prediction across many teams, leagues, countries, and seasons with high accuracy. The problem and objective idea was heavily inspired by the dataset you sent – it is a really good data.

**Solution:**

The solution involves developing machine learning algorithms that can achieve great performance on the aim and objective of the paper. This algorithm must have similar performance across the teams, leagues, seasons, etc. This would show that it is possible to develop this kind of system using machine learning.

**Method:**

We already had machine learning and deep learning methods in mind but it was for a better story telling that I made it look as though the deep learning models were used after the machine learning models didn’t perform as hoped.

First, I tried 5 machine learning model. So, I replaced the Gaussian Mixture Model (GMM) with another clustering classification model, K Nearest Neighbors (KNN), because it didn’t perform well or rather it was not suited for this task. You used Logistic Regression (LR) which I also used. The other ones that were added (Naïve Bayes (NB), Support Vector Machine (SVM), and Random Forest (RF)) are very popular models that have been used a lot!!!

**K Nearest Neighbors (KNN):**

This algorithm works by mapping each data point in the dataset into a feature space based on the values of their features. It then assumes that data points that are close to each other in the feature space have similar features and hence belong to the same class/label and those of farther distance are less similar and are of a different class/label. During prediction, this algorithm maps the new data into the learned feature space and locates the K-nearest training data points according to a distance metric. The class label of the new data is then determined based on the majority of the labels from the K-nearest data points.

**Naïve Bayes (NB):**

Multinomial Naïve Bayes algorithm is based on the Bayes theorem. It follows that if a feature (e.g. home team) has a certain value (e.g. Chelsea) what is the probability that it belongs to each of the defined classes (home wins, away wins, and draw). This probability is then calculated for all features and combined. The class with the highest overall probability is then taken as the algorithm’s prediction.

**Support Vector Machine (SVM):**

This algorithm is trained to find an optimal decision boundary (called a hyper-plane) that best separates the training data into two or more classes, such that the distance between the training data points to the hyper-plane (called the margin) is maximized. I used a linear SVM, so the decision boundary here would be a straight line. Keep in mind, however, that the data is of high dimensions, so yes, a straight line can do a great job at dividing it. In fact, the higher the dimension, the better the performance of a linear SVM.

**Random Forest (RF):**

Random Forest algorithm is an ensemble of a large number of small decision trees (called estimators). Each estimator is trained on a subset of the training data and on some of the features to ensure they learn different things. This algorithm then takes the prediction of each estimator and chooses the label predicted the most number of times as its prediction. Random forest generally outperforms using a single decision tree.

**Logistic Regression (LR):**

Logistic Regression algorithm is based on the popular linear equation, y = mx + c, where m is the weight, x is the input, c is the bias, and y is the output class. The major difference is the introduction of an activation function (e.g. ReLU, sigmoid). The job of an activation function is to ensure that the model can learn data patterns that are not linearly separable and hence complex data.

**Dense Neural Network (DNN):**

The success of the RF algorithm helped me in deciding to use an ensemble deep learning model.

The Dense Neural Network model or deep learning models in general are based on Neural Networks (NNs) which were inspired by the neural networks of the human brain. Neural networks are made of layers, each layers are made of neurons, each neurons can be thought of as exactly a LR algorithm. So, neural networks can be thought of as consisting of several LR algorithms. DNNs are called dense (and sometimes fully-connected (FC)) NN because they connect each neuron in a layer with ALL of the output from the previous layers. Unlike DNNs, other NNs like CNN usually connect parts and sections.

**Embeddings:**

Now, most machine learning models don’t work with categorical data, so this kind of data has to be converted to numbers. Converting them to unique integers is not enough and causes a misinterpretation which I’d explain. Say there are three values that a categorical feature called fruit can take, (orange, banana, and pineapple). Now if you converted them to integers, (1, 2, 3) the model would interpret it to mean a banana is 2 times an orange, and a pineapple is 3 times an orange which does not make any sense.

One-hot encoding comes to the rescue and now the model knows they are different and not multiples of each other. Embeddings however takes it to the next level. What if you could learn numerical features about these fruits that would help you easily understand how they affect the output label? Depending on the task, numerical features like price ratio, rarity, popularity, etc. can be learned from the data without you explicitly teaching the model. Embedding layers usually shine a lot in text related tasks like text generation where there could be over 10,000 possible value of a feature. One-hot encoding would give you an array of size 10,000 whereas an embedding size of 516 could be used to adequately understand each categorical value.

**Method explanation:**

So, four DNN models were tried: plain DNN (one-hot), plain DNN (embedding), ensemble DNN (one-hot) and ensemble DNN (embedding).

Both embedding models outperformed the one-hot models even while reducing the input size by over 1000. Interesting things would be found in the embedding that was learned but that requires more time to investigate and is not directly related to the task (miscellaneous analysis).

Both ensemble models outperformed the plain model they were based on. The reason why ensemble methods generally perform better than their individual model is because they have different views and perspectives on the task. Remember how the RF algorithm was implemented. Each sub-model was trained on part data and part features. This same method was applied to the ensemble DNN models and the reason for doing that is if they were trained the same way, then they’ll likely end up with the same weights and biases. And, eventually their combined models would produce not better but the same accuracy.

**Questions:**

1. Sensors are technologies that are used for monitoring, tracking, and data collection purposes. The goal-line technology is an example which monitors when the ball crosses the goal-line and becomes a goal.
2. You sent me the link (<https://football-data.co.uk>). I asked about where you got the data and you sent this link.
3. You split the data into training and testing data because if you don’t, you would not know how well the model performs on unseen data. The goal of developing a model is to be able to use it to predict the outcome of some new/fresh real-life data. So the training data is used for training the model, while the testing data is to really determine how well the model performs on data it has not seen before i.e. data it was not trained on.
4. Imagine training a dog to always fetch a stick when you throw it. What happens when you throw a ball instead? Did the dog learn to fetch whatever you throw, only sticks thrown by you, only sticks thrown by anybody, etc. In this case the object thrown, the person throwing it, is the input, while fetch or not fetch is the output. Without a different set of data, you cannot properly evaluate if the model as truly learnt what you want it to. You can say, oh why not train it on balls too; the problem is that you cannot always account for all possible input. I mean what if I throw a shoe, refuse, water, etc. the possibilities are quite much.

The model is also testing on the training data to see how effective it is; if at all it is suitable for the task. Using the dog example, when you throw a stick does the dog always fetch it? Now, of course dogs have mind of their own and are not robots, but if a model is not even good at what it was trained on then you can’t expect a good performance on what it was not trained on (testing data).

1. Epoch means the number of times you have trained your model on your ENTIRE dataset. Usually you train you model repeatedly on your data to get a decreasing loss value. You can think of the loss as the objective of the task. For example, say you are predicting the price of a product based on some factors. Now, you would want the model to predict values that are as close as possible to the actual true value. The loss in this case could be something like this: loss = {true value} – {predicted value}. Essentially, you are saying I want the difference between the true value and the predicted value to be reduced as much as possible (remember that the model is trained to reduce the loss not increase).
2. I first explain each metric. They are all used in evaluating the performance of a model.
   * Accuracy is how well the model was able to correctly predict the label of the data,
   * Precision measures whenever the model predicts a particular class (e.g. home wins) how many times was it correct.
   * Recall measures of all the time a particular class (e.g. away wins) appears in the testing data, how many did it correctly predict.
   * F1-score measures like an average of the precision and recall score. It favors a model has similar precision and recall VS one where the precision and recall are not within close range. In fact, a precision of 100% and recall of 0% would give 0% F1-score whereas, 30% precision and 30% recall would give around 30% F1-score.

Accuracy score are known to not be very good measure of performance. Think for example a model that predicts if a coming asteroid that is as big as a football’s ball would enter into earth’s atmosphere. While it is possible, it is quite rare and predicting “No” all the time can easily get you more than 90% accuracy even though you failed at predicting when it is “Yes” which is more important.

Precision measure of all the times you predicted “Yes” how many times was there actually a colliding asteroid. Recall measures of all the time there were actually a colliding asteroid how many did you predict, This individual metrics also have their downfall. Predicting “Yes” all the time can get you 100% recall and predicting “Yes” only when you are 100% sure it would collide can get you 100% precision. However, you cannot outsmart both metric at the same time.

This is where F1-score comes in. As explained in its definition, you have to perform well on both precision and recall to be able to get a high F1-score.

1. This has been explained in the Embedding section.
2. ReLU has been explained under Logistic Regression section. ReLU is a type of activation function, and what they do is to help the model understand patterns in the data that are not linear i.e. patterns that are formed through a complex relationship between features. For example, the year of release of a phone and the quality of the camera a linearly proportional to the price of the phone (recent release, better camera, equals more expensive phone). The success of a movie in the cinemas however does not linearly depend on the number of star actors/actresses, or the length of the movie. Although, they are important for predicting the success, more stars do not mean more success and vice versa. With an activation function you can better understand these non-linear patterns.

Optimizers (e.g. Adam optimizer) are used for updating the weights and biases of the model (remember m and c from y = mx +c). To make the model better at predicting y, you have to keep updating m and c. So the optimizer takes the gradient (i.e. the changes it should make to get a more correct prediction) for both weight and bias, and then does some smoothening to it. The reason for this is to ensure the model learns an optimal solution rather than a solution that works for just one example.

1. Estimator is just the word used to refer to the individual models within an ensemble model.
2. I’ve talked about hidden layers in DNN section. Neural networks (including DNNs) are made up of layers. The popular intuition about hidden layers is that early layers learn abstract things while later layers learn task specific things. Take human face detection for example. The early layers are thought of as learning things like lines, curves, shapes, while later layers put these things together to learn eyes, mouth, ears, etc. The hidden layer basically is everything in-between the input layer/data and the output layer/prediction. After all, cats and dogs also have eyes and ears. The output layer would be responsible for taking this features and predicting, “yeah this all looks like that of a human, human face detected” or “nah, I did not see any eyes or this eayes are for cats”, etc.

If you remembered under DNN, a DNN is made of a lowest unit called neurons which are exactly like logistic regression. Then this neurons forms layers and layers form the entire model. The hidden size then is the number of neurons in a particular hidden layer (or any layer). Different hidden layers can have different hidden size, but in the paper I used one hidden layer, so I reported one hidden size.

If this was not enough or you have some questions, feel free to reach out to me.

Thanks.

Banjoko Judah.