SVM with iRace tuning Summary

Table of Contents:

High Level Summary:

Using SVM models, we can correctly predict whether an NFL games over will hit 58.75% of the time. Using Classification thresholds along with the model, we can correctly predict 72.41% of outcomes (whether the over will hit or not).

The most important indicators & their directional impacts were:
(The inverse would be true if we were discussing "the under")

- Average Points scored by the Underdog at Home
  - Once this surpassed 20, the chances of the over hitting began to increase
  - Anything over 30 or below 10 resulted in a higher probability that the over will hit. (Think getting blown out vs blowing the other team out)
- Average Points scored by the Underdog
  - A score of around 15 yielded the lowest probability of the over hitting, and as this increased to 20,25,30,40 – the probability of the over hitting increased as well.
- Average Points surrendered by the Underdog
  - Very low scores & very high scored resulted in a higher probability of the over hitting.
  - Median scores around 30 resulted in the lowest returned probability
    - This can be explained by teams who either get blown out (score very low & give up large scores) or do the blowing out (score very high & not give up many points) in which both cases would result in the over hitting.
- Home offense overall rating & Favorite Average Points Scored both followed the logic outlined above & were also among the more important features.
- Over/Under Line
  - As the over/under line set by Las Vegas increased, the probability of the over hitting decreases. This is as one would expect.

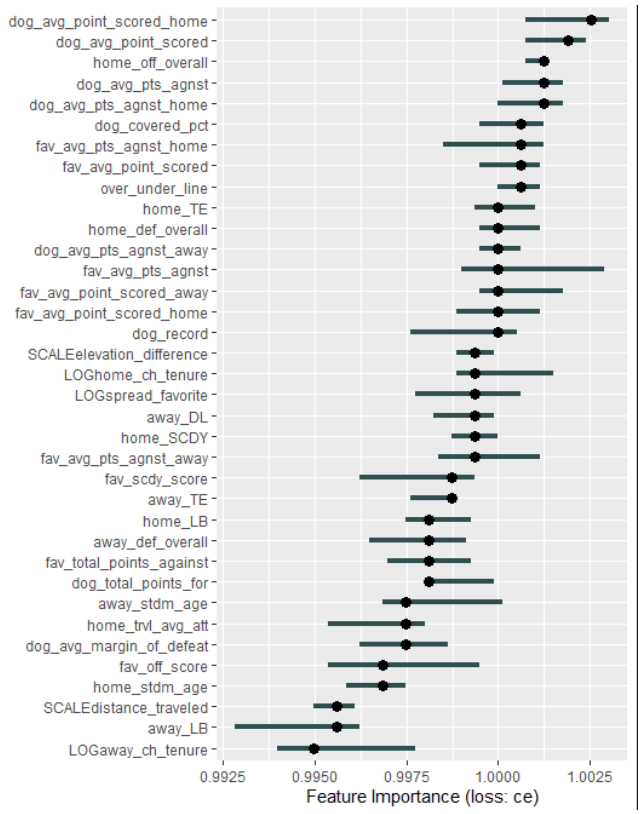
Using classification thresholds decreased the amount of bettable instances (since there was essentially a cutoff point), but it helped increase the accuracy.
For our case, this is a trade off we would make since it is not a requirement to place bets on every game & it is even more beneficial to parlay bets together.

The "over" in a game is the combined total points for a given game. Sports Books set an "over" & bettors can choose to bet the "under" or the "over"

Quick example:

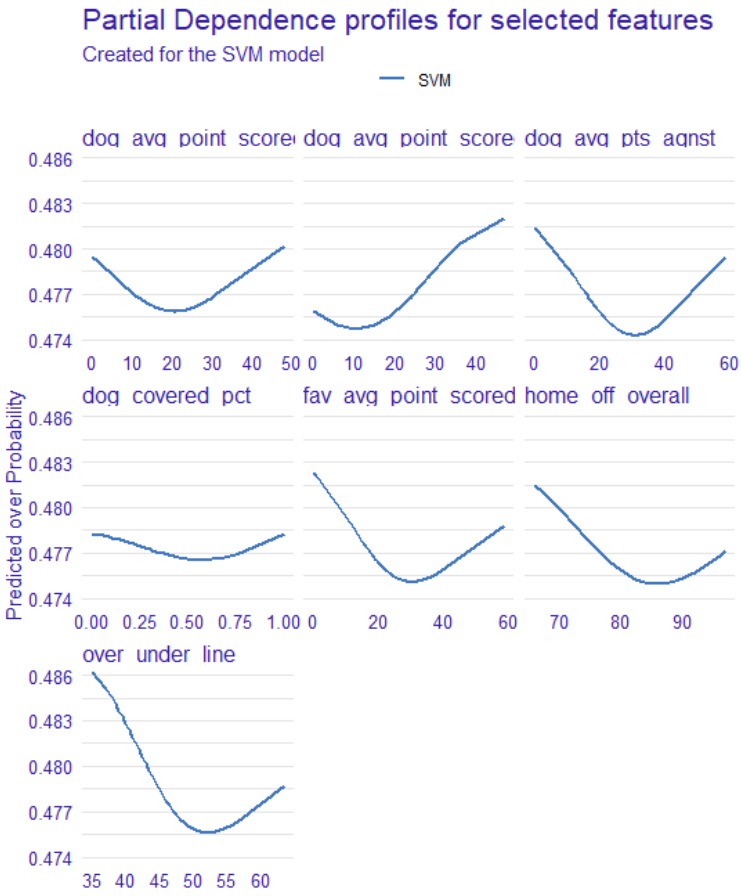
- The over/under line is set at 42.5
- The game ends 20-21
  - Total of 41 points
- So, in this case, the over did not hit.

Feature Importance:



We can see that features which included any kind of point accumulation were among the more important features. This supports expectations that offensive & defensive behaviors are the driving forces behind the over/under outcome. Next came variables that primarily consisted of team skill/personnel. Such as position ratings. This indicated that teams with different strengths would have different expected over/under outcomes – however, importance plots do not tell us the directionality of the feature. We can use PDPs and SHAP plots to investigate further

Partial Dependency Plots:

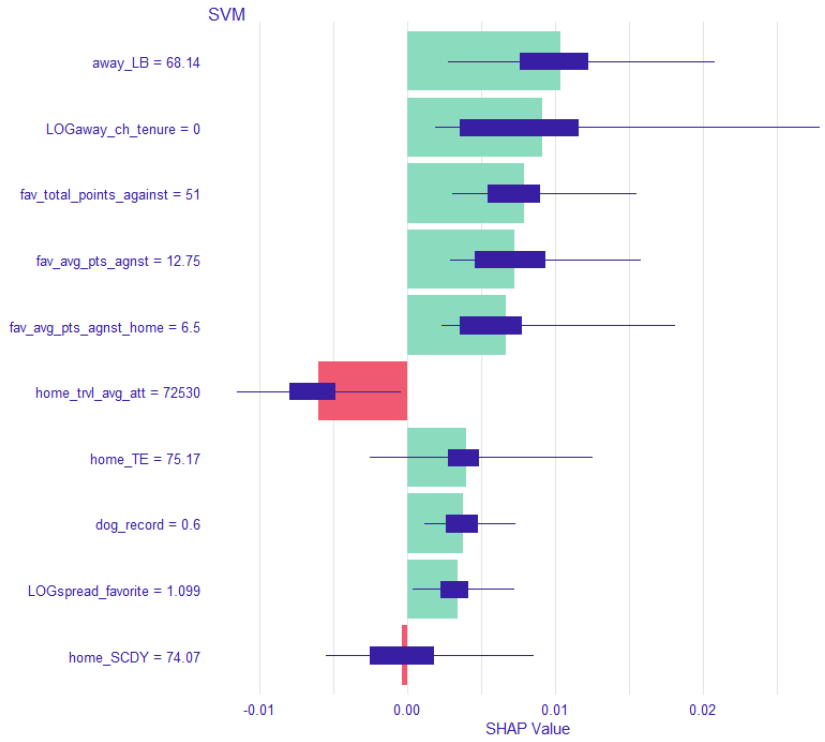


The PDPs can help show the relationships between certain features & the probability that an outcome will be predicted “all else equal”. As discussed above, there is a main “theme” we are seeing here that has more value than individually picking through each variable & its over/under relationship.
The main theme we are seeing is that there is this repeating “U-shape” which illustrates a paradox that tells us, if a team scores a lot or a little, then there is a greater chance the over will hit versus if a team just scores in the “middle” range.

So why this this? And why is this a paradox?
For one, you would expect there to be a positive directional relationship between average points scored (and surrendered) with the over hitting. Which there is. I.E., the more points, the higher chances the over will hit.
But on the other hand, when there is 0-15 points scored/surrendered, there is a higher probability the over hits than if there was 20-30 points scored/surrendered.
We have already touched on this, but to reiterate the point – it has to do with either “getting blown out” or “blowing out” the other team.
If the over/under is 44.5 points. Team X scores 13 points Team Y scores 42 points. The over hit, and this can help explain the U-Shape.
Now if Team X scored 20 & Team Y scored 24 – the over did not hit.

This is how it would happen, but why is this occurring?
In a close game, teams are going to play to the final whistle & try not to give up any more points because they might lose.
In a “blow-out” teams tend to not play as hard at the end & often times give up “garbage time” scores where one team may surrender an easy score since the game is already essentially over.

SHAP Plots



SHAP plots help us not only determine the directional impact of a variable, but also help determine the intensity. SHAP plots over show the features value (high or low) with the bar colors & then show the impact based on the direction & size of the bar.

We can see which features have more positive impacts with higher values (Coach Tenure, Favorite Total/Average Points Against, Underdog record, Spread, etc)

Features such as Travel Attendance for the home team & Home Team Secondary have negative impacts with lower values.

This is not really the best SHAP plots package, most of them have a color scale & not just green/red so you can also see the feature value extremities.

Overall, in my opinion, the PDPs helped us best understand what is going on for this specific model.

In Depth Explanation:

SVMs

Preprocessing:
- Loaded Packages & Cleaned Data

Variable Selection:
- Used a random forest model on the training data to reduce the feature
  - OOB error for each data point is recorded
    - Prediction error via bagging
    - Bagging: Subsamples with replacement creates the training samples
    - The model is trained on the bagged observations
    - The OOB error is calculated on the observations OOB
    - The selected feature values are permuted for the training data (noise/randomness is added) then the OOB error is computed again
    - The difference between the OOB error before & after the permutation is calculated & this is how importance is determined
      - Mean Decrease Accuracy (MDA)
        - How much the accuracy decreases after the process above
      - Mean Decrease Gini/Impurity (MDG)
        - How much the node purity decreases after the process above
    - Kept variables with an MDA > 0 & an MDG > 5

Preprocessing 2:
- Had to convert data all to numeric values for the SVM model

MLR3 Package:
- Set the learning tasks
- Set the learner (classif.svm)
- Set the tuning parameter bounds
- Set the evaluation metric (logloss)
- Set the resampling technique (Repeated CV (3 repeats & 10 folds))
- Set tuning budget (200 evals)
- Set the tuning method (iRace)

Selected Parameters:
- Kernel: Radial
- Gamma: 0.0944
- Epsilon: 0.0716
- Cost: 0.9421
- Tolerance: 0.0031

SVM Summary: (*Support-Vector Networks, VAPNIK, CORTES (1995))*

Background:
- Originally developed as a learning machine for binary classification.
  - Input vectors are non-linearly mapped to a high-dimension feature space & in this feature space a decision surface is constructed (HyperPlane/Surface)
  - The terms “hyperplanes” vs “decision boundaries”
    - A hyperplane is a decision boundary for a linear SVM
    - A non-linear hypersurface is a decision boundary for a non-linear SVM
      - I.E., when using kernel functions (which we are), the decision boundary will no longer be a straight line (plane).
- Support vectors are the data points which lie closest to the hypersurface
  - Support vectors are the most difficult data points to classify & lie on the “margin” surrounding the hypersurface
- SVMs attempt to maximize the space/margin between the support vectors and the hypersurface & the decision function becomes specified by the support vectors

Main Ideas SVMs, specifically Radial Kerels

- Kernels project the data into a higher dimensional space
  - We are trying to get the transformed inner product of the original data through a kernel function
- Specifically, for a radial kernel – the goal is to consider the infinite number of interactions between features
  - Since radial kernels are considering an infinite space, we have to first take the original inner products & then use the kernel function to get the transformed inner product
  - The radial basis function kernel is $K(X_1,X_2) = e(-y||X_1-X_2||^2)$
    - We can express this function only in terms of the inner products of the original data
    - And eventually, we are able to consider all infinite interactions without ever having to do an infinite dimensional projection
- In summary, the radial kernel allows us to consider an infinite number of interactions
  - Once this is done, we can form a decision boundary based on the results of the radial kernel & then optimize the decision boundary
- The decision boundary hypersurface which separates the two classes
  - Decision Boundary: H: $w^t\Phi(x)+b = 0$
    - And the surfaces which touch the support vectors can be defined as
      - H1: $w^t\Phi(x)+b = +1$
      - H2: $w^t\Phi(x)+b = -1$
  - We measure the distance from a hypersurface to a point vector as
    - $d_h(w^t\Phi(x_0)) = (w^t\Phi(x_0)+b)/(||w||_2)$
  - Keeping in mind the goal is to maximize the space/margin between the support vectors and the hypersurface
    - $w^* = \text{argmax}[\min d_h(w^t\Phi(x_0))]$
    - This represents the distance of the closest point to H (the hypersurface)
    - To simplify/scale the distance, we can set this "distance of the closest point" to be 1.
      - So that, $[\min d_h(w^t\Phi(x_0)+b)] = 1$
        - Since there will be error in our model, we introduce a new variable (cost) which allows for wiggle room (Cost = $\sum\xi_n$)
        - $d_h(w^t\Phi(x_0)+b) \Rightarrow 1-\xi_t$
        - Now, we still have the Φ term which is difficult to compute
          - (Highly Dimensional projections with infinite interactions which will be covered shortly)
    - Now taking a step back to before we introduced the cost, we can then convert it to minimization problem: $d_h(w^t\Phi(x_0)+b) \Rightarrow 1$
    - Or $\min(1/2)||w||^2$ subject to the above $d_h(w^t\Phi(x_0)+b) \Rightarrow 1$
      - To solve the problem above, we can use Lagrange Multipliers & kernelization
      - Which, through some derivations, results in no Φ term. Meaning we do not need a complex basis to model highly dimensional data – which is why modeling highly dimensional interactions can be achieved without ever actually having to do highly dimensional projections
        - This can be achieved through kernel functions as presented at the beginning of this section
        - (Highly Dimensional projections with infinite interactions)

Our selected parameters:

Kernel: Radial

- Known as a Radial Basis Function/Gaussian Kernel
- $K(X_1,X_2) = \exp(-y||X_1-X_2||^2) = \exp(-y\sum(X_1-X_2)^2)$
- Also written as exp(– γ|u – v| 2) in the package documentation
    - $||X_1-X_2||$ is the Euclidean distance between X_1 & X_2
        - Since we are squaring it, it becomes the squared Euclidean distance

Gamma: 0.0944

- y is the gamma parameter which accounts for the variance & smoothness of the hypersurface (larger = high variance, smaller = low variance)
    - It essentially scales the squared distance between X_1 & X_2
    - And controls the distance of the influence of a single training point
- So, for a SV classifier, the equation becomes (the decision function)
    - $f(x) = \beta_0 + \sum\alpha_i K(X_1,X_2)$
        - αi is a weight parameter (Lagrangian multiplier)

Epsilon: 0.0716

- Epsilon is a parameter which specifies a margin of tolerance around the "true value" of a sample, so that a sample within a certain range is still considered "correct"
    - Where the term "correct" can be interpreted as an instance where no penalty is given to the associated error.
    - Epsilon impacts the tolerance of errors, meaning that there is a "balance" to find between penalizing everything & penalizing nothing.

Tolerance: 0.0031

- The tolerance parameter controls the termination criterion which is a setting for stopping during optimization. Lower values require more optimization & higher values require less optimization
    - It is essentially providing "wiggle room" to allow support vectors on either side of the hypersurface to be equal to 0, if they are within the tolerance range (*pythonprogramming*)
        - So, with lower tolerance, less space is granted, resulting in a more stringent/optimized surface

Cost: 0.9421

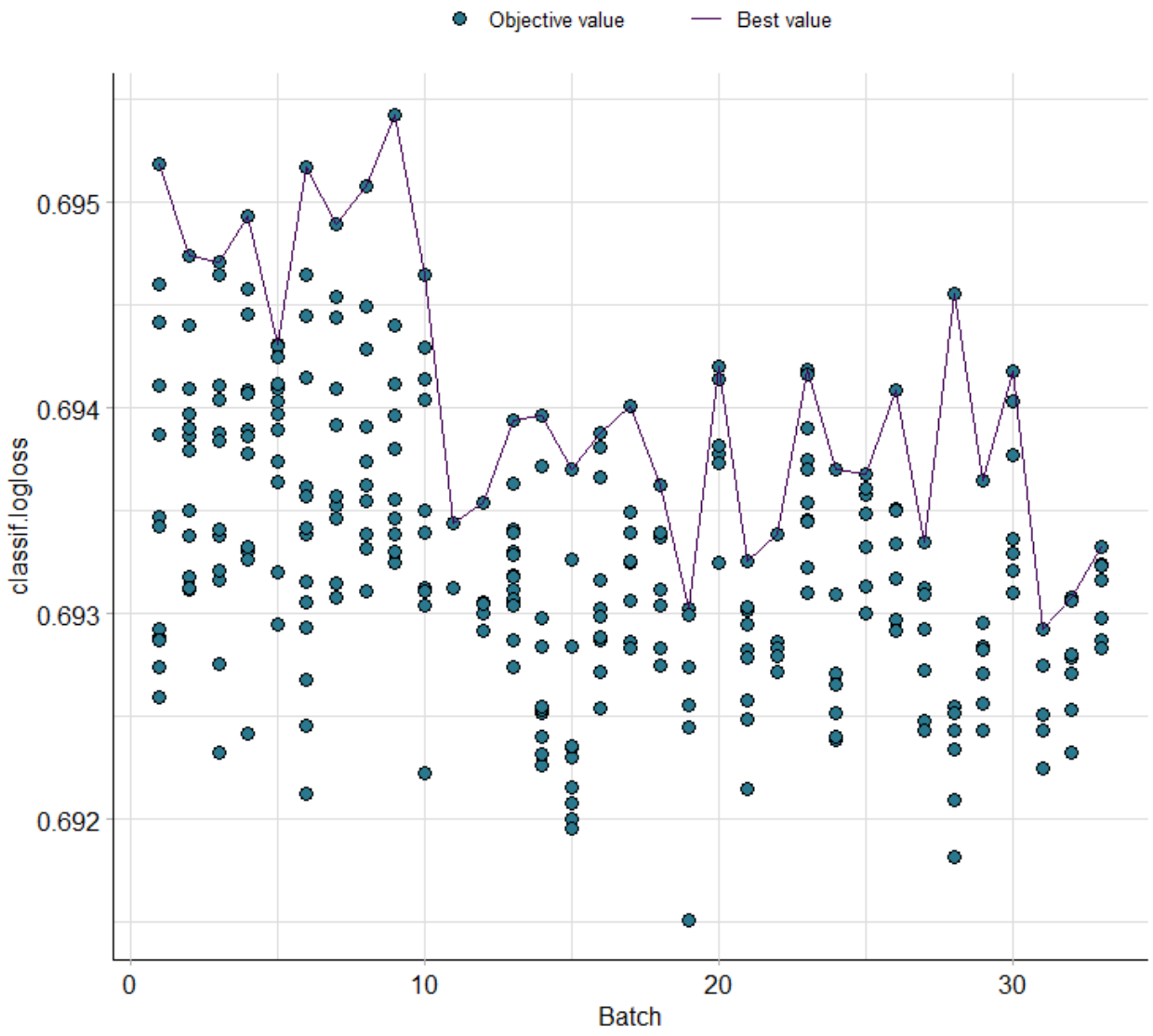
- A parameter which helps control the overall fit & error penalization.
- It is similar to tolerance, discussed above. And was introduced in the main ideas section above.
    - But basically, the Cost parameter helps control the regularization of the loss term. It helps determine how aggressive the model is regarding misclassifications.
        - A lower Cost, the stronger the regularization – so the model will be more tolerant to misclassifications and more general in a sense.
            - As it approaches 0 the decision boundary becomes more linear
        - The higher the Cost, the plane will be more aggressive, resulting in a smaller margin – which may lead to overfitting.

References:

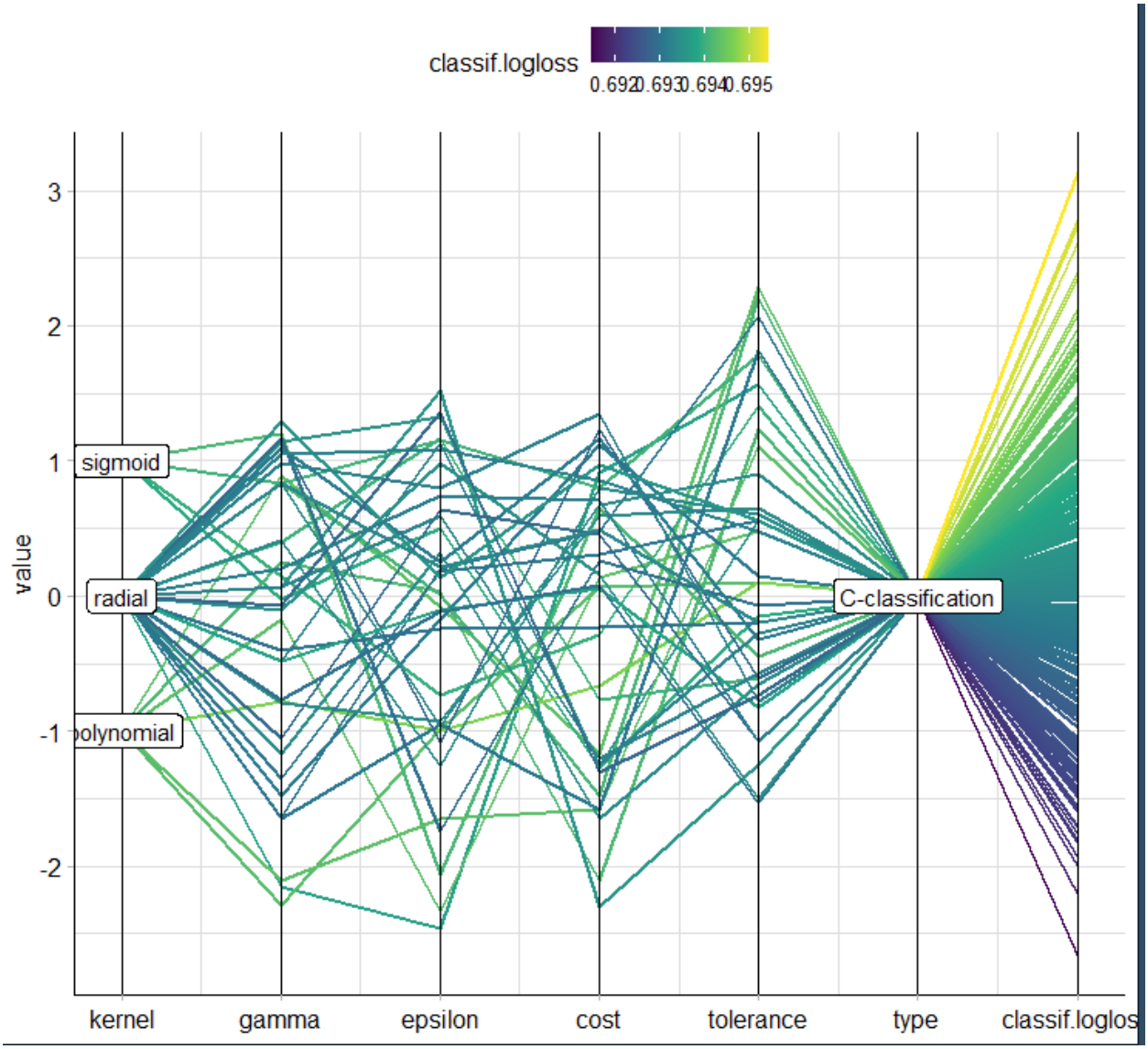
(*Support-Vector Networks, VAPNIK, CORTES (1995)*)
(School Notes (University of Arkansas Economic Analytics))
(Towards DataScience)
(ritvikmath)
(CodeEmporium)
(mit.edu)

The tuning method (iRace) is covered on the last page of the summary.
This methodology was outlined in the GLM summary, so including it in the body of this summary would be repetitive.
It is on the last page of this document,

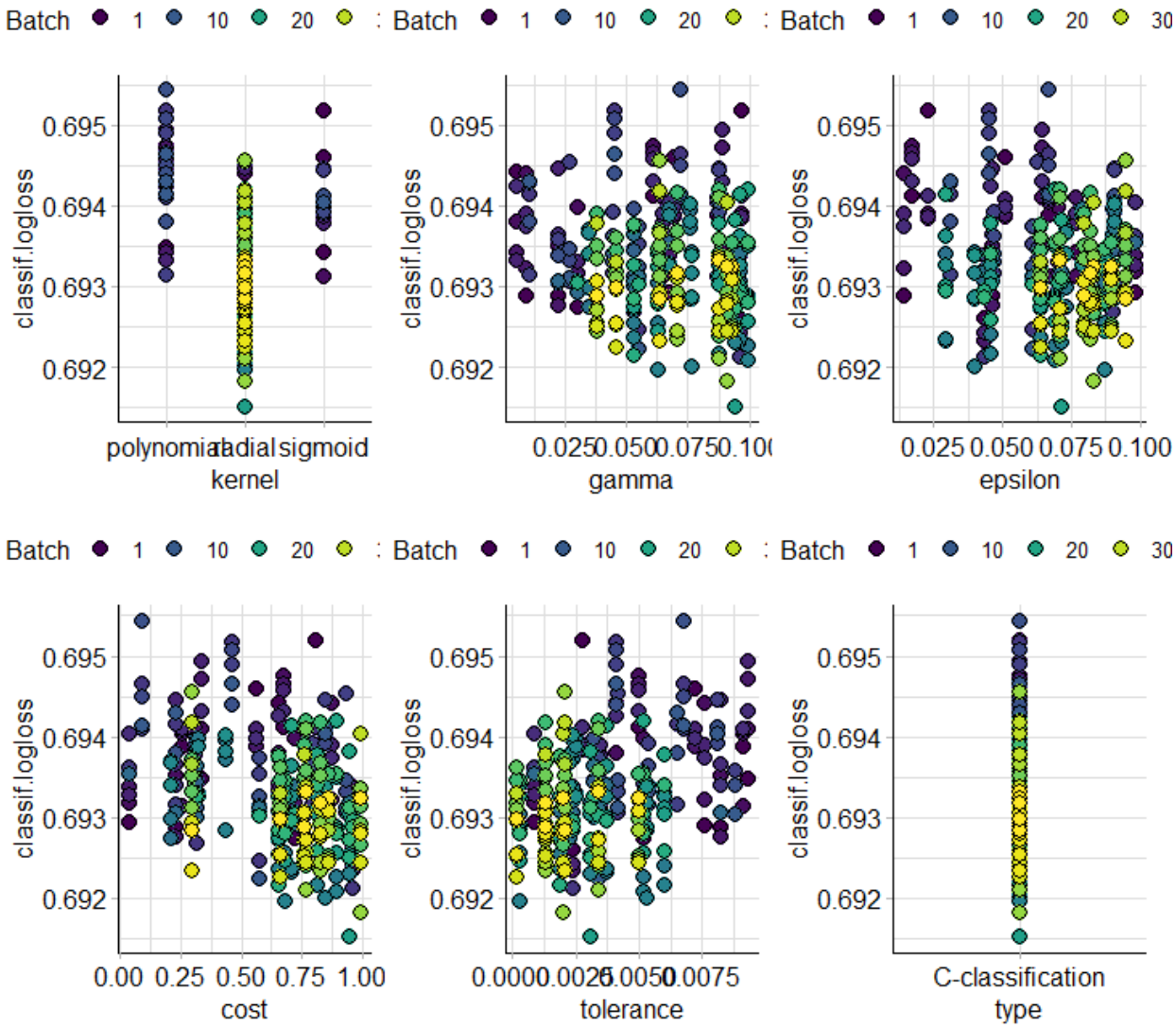
Now that we have discussed the model and the parameters, we can visualize the tuning process below



As expected, with more tuning – the model's performance was generally increasing.



We can see that the radial kernel, with a higher gamma, moderate epsilon, higher cost, and lower tolerance provided the lower log loss values.



Individually viewing the parameters:

- Radial Kernel
- Higher Gamma
- Moderate epsilon
- Higher cost
- Lower Tolerance

Tuner: iRace (Iterated Racing)

- Could have used other methods such as random search/grid search
- Chose Iterated Racing

Background:

References:

- *"The irace package: Iterated racing for automatic algorithm configuration"*
  - *(López-Ibáñez, Dubois-Lacoste, et al. 2016)*
- *"A Racing Algorithm for Configuring Metaheuristics"*
  - (*Birattari, Stutzle, et al. 2002)*

- Tuning used to be done in an ad-hoc fashion
  - This had many drawbacks
- Automatic algorithm configuration was developed
  - The goal was to find configurations that minimize some cost measure
  - The goal was also to find the configurations that generalize to similar, but unseen instances
  - There are many methods for automatic algorithm configuration (tuning)
  - One of these methods was Iterated Racing
    - Racing: Selects one configuration among several candidates using sequential statistics
      - Candidates being parameters.
      - I.E. quickly reducing the number of candidates & focusing on more promising candidates through statistically guided experiments, while minimizing the number of experiments
      - Dropping inferior candidates speeds up the procedures
    - iRace implements the I/F Race Algorithm (Second Reference)
    - The I/F Race Algorithm is based on the Freidman Test as the statistical method for hypothesis testing & ranking of the candidates
      - With respect to the Friedman Test, in the case of I/F Race testing…
        - The null hypothesis is that all possible rankings of each candidate within each block are equally likely for a given iteration
          - If the null is not rejected (the rankings are equally likely at some significance level[1]), all the candidates tested will be passed to the next iteration
          - If the null is rejected & the candidate ranks are not equally likely, pairwise comparisons are executed between the best candidate & all the other candidates in that iteration – all candidates that are significantly worse than the best is dropped & will not appear in the next iteration.
      - This is essentially how irace tuning works. It gets the name since the candidates are "racing" against each other & the winners are the only candidates passed on to the next iteration.

- [1] Along with the rankings being equally likely, T has to be approximately X^2 distributed with n-1 degrees of freedom. Defining T & X, along with the equation required to evaluate the pairwise comparisons of the candidates would have dramatically increased the complexity & scope of this summary. Both equations are in the paper referenced. *"A Racing Algorithm for Configuring Metaheuristics"*