PART II

The following may be unnecessarily long but it is hopefully illustrative of my thought process!

- Here our task is to determine whether or not we think the features contained in the "dataUS.csv" dataset have any predictive value when it comes to forecasting the associated forward returns of the underlying instruments, given in the "MLTestRiskDataUS.csv" dataset.
- We've got a set of heterogeneous features (categorical sector info, discrete clusters "ds", continuous "short"/"price" values, etc.), along with missing values here and there and some indicators for special cases (e.g. the "Short" feature is marked with "New" instead of a numerical value when an instrument first starts trading, presumably).
- My default modeling choice in these cases – barring some strong prior about the mathematical relationship – is to use a tree-based model. On the other hand, I tend to think that simpler linear models not only serve as important benchmarks, but that added complexity requires justification. (Although in this case, we're not interested in the tree based model for the complexity/capacity, we're more interested in it for the elegant handling of missing or mixed-type data.)
- For linear models on noisy financial data I'll nearly always use regularized and/or robust methods including ridge, lasso, and support vector regression (SVR). I'll generally use ridge in place of OLS, and lasso when I suspect some features are unimportant or there is some other problem-specific reason that sparsity is desired. For very noisy data the properties of the SVR loss function are desirable (ignores errors within ε of target, linear outside that region and therefore somewhat robust to outliers).
- So for this problem I decided to pursue two models: a gradient-boosted regressor (using XGBoost) and a linear support vector regressor (using scikit-learn).
- **Data**
    - We've got missing values of the "Short" and "Price" features, as well as some categorical indications here (e.g. when an asset is marked "new" or when the price of an asset is marked "top").
    - For a tree-based model which can partition the input space, we could do something like fix arbitrary values for these indicators, e.g. set the "Price" feature to 999% when we see "top," or set the "Short" value to -999% when we see "new." The model can learn to separate the regions of the input space and recognize these values as being distinct, and we don't need to enforce any of our own biases by choosing how to set these values or impute missing ones.
    - For the SVR, we should instead do some kind of imputation (as well as standardize the numerical features and one-hot-encode the categorical features). For "new" and or missing values, it seems to make sense to set the values of "Short" and "Price" to 0%. The mean of the numerical values for these features is 0 to well within the variance of the data, and this is sort of an intuitive choice to begin with.

- As for “top,” this appears to occur only 22 times (out of 33600) in the dataset. Given that I’m not totally sure what is meant by this value, I’ll simply set it to the mean of the non-missing values (which again happens to be essentially 0).
- We might be worried about mean imputation given that there could be outliers (in which case perhaps we’d use the median), but these features don’t appear to be too badly affected by this issue.
- Finally, there are just a few typos, e.g. we see a value of “-10.1% A” which we change to “-10.1%”, etc. (Unless this is a notation indicating something specific which I’m not understanding; regardless, it’s rare).
- We’ll then want to convert the percentage values (which so far are strings) into floating point numbers. We’ll turn the sector feature into a one-hot encoding indicating when an instrument belongs to a specific sector class.
- For the SVR we’ll go ahead and standardize the numerical columns of data.

- **Models**
  - We’ve now got to read in the relevant bits of the “MLTestRiskDataUS.csv” dataset (the forward returns), and align them correctly with the corresponding “id” and “pk” rows.
  - We’ll need to make some hyperparameter choices for each model, which we’d generally do via cross-validation.
  - In many financial time series prediction tasks, cross-validation is a tricky exercise. If we think the effect we’re after is nonstationary or changing in time, then we’d want to construct our CV folds so that the model is always trained on data which precedes the validation sets (we always maintain temporal ordering of our folds). In this case, I don’t have explicit time ordering (unless that’s what “pk” is indirectly indicating). But more importantly, if we suspect the info in this dataset has predictive value (imagining that this is some alternative dataset or some set of sentiment indicators from a vendor), then it ought to have value which is sort of time-invariant. Therefore, we’ll do cross-validation in the traditional format.
    - We won’t do a brute-force grid search in order to minimize a regression metric. We’re also concerned with classification accuracy (making a correct directional forecast) as well as some relative long/short balance in the final predictions. We don’t want seemingly high-performing model which always goes long (or vice versa).
  - I chose to predict 5 day forward returns. While one can get smaller regression errors on the 1-day and 3-day returns, this is partly due to the fact that the variance in these returns is smaller (there is more vol over longer periods). Put another way, the R^2 is not actually better for these shorter periods. Also, we seem to be able to get better accuracy on the 5-day returns (as measured by the percentage of time we correctly forecast the sign of the forward move). Finally, a consideration in evaluating an alternative dataset is the alpha decay of the data, so it’s desirable if the data appears to be predictive of returns over longer horizons.

- While the linear SVR appeared to work about as well as the XGBoost model as measured by the usual regression metrics, I found it troubling that the model seemed to predict predominantly negative returns. The dataset is not badly imbalanced negative vs. positive, but the predictions were imbalanced. At first I wondered if this was due to the intercept term, but it persisted even after enforcing the intercept vanish.
- I observed the same effect when fitting shallow XGBoost models. I would have liked to spend more time trying to understand the source of this, but simply had to wrap this up at some point!
- I found that I needed to use deeper trees (more flexible models) in order to achieve some reasonable balance in the sign of the predictions. As I increased the depth I also increased the level of L2 regularization on the model weights. This seemed to be a nice tradeoff between goodness-of-fit (again, as measured by cross-validated regression/classification metrics) and more evenly balanced predictions. So, in the end my final predictions were generated with a relatively deep but highly regularized XGBoost tree-based regressor.
- In practice, I might generate final predictions by doing an ensemble average over separate sub-models with different (reasonable) hyperparameter choices, but I decided that was beyond the scope of this project!