III. Model documentation and write-up

Information included in this section may be shared publicly with challenge results. You can respond to these questions in an e-mail or as an attached file. Please number your responses.

Who are you (mini-bio) and what do you do professionally?

Yanick Medina (@camaron_ai)

My name is Yanick Medina, I'm twenty one years old and from Venezuela. I am currently pursuing my undergraduate degree in Electronic Engineering. I achieved the third place in the rodeo II sprint #1 competition and ended up in the first 7% of the M5 forecasting competition. I am passionate about programming and always willing to learn more about ways to solve real-world problems using efficient algorithms and data science.

Hamlet Medina (@NataliaLaDelMar)

I am Hamlet and work as a Chief Data scientist at Criteo in Paris. I hold two master's degrees on Mathematics and Machine learning from Pierre and Marie Curie University, and a PhD in Applied Mathematics from Paris-Sud University in France, where I focused on Statistical Signal Processing and Machine Learning. Before my graduate studies, I worked in Control Systems for Petróleos de Venezuela.

What motivated you to compete in this challenge?

We decided to participate in this challenge because only the idea of building a system that can predict the Disturbance Storm-Time Index in real-time using measurements collected from two satellites is extremely cool. Before starting the challenge, we did not appreciate the huge impact that Solar-Wind has on the Earth Magnetic Field. With no previous domain knowledge on the topic, we were excited to use Machine Learning to see what we could do. We are always looking forward to solve and learn more about real problem with a potential social impact.

High level summary of your approach: what did you do and why?

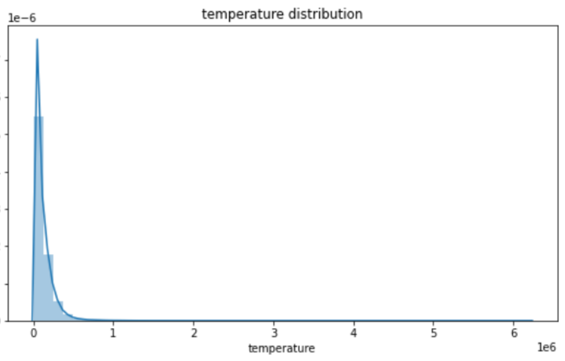
Our solution is decomposed in the following steps:

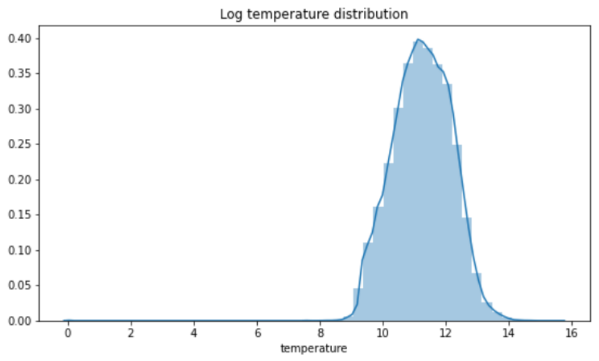
Feature Engineering

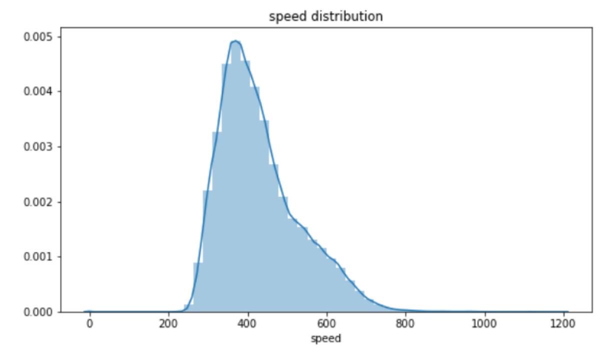
The solar wind data provides information about the Interplanetary-magnetic-field using 2 different coordinate systems. Particularly, we do not use the time series using the GSE coordinates. Our solution uses the following time series for modeling

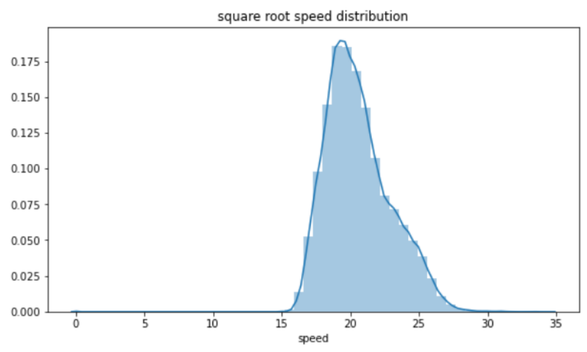
- bx_gsm
- by_gsm
- bz_gsm
- theta_gsm
- bt
- density
- speed
- temperature

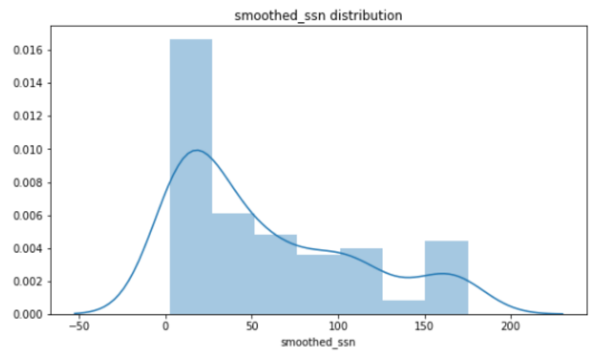
Before applying any feature engineering preprocessing, we studied the distribution of the temperature, speed and smoothed_ssn time series. We noticed some degree of skewness on them. Skewness normally hurts modeling, especially in models that assume normality in output distributions. In order to reduce it, we apply the log and square root transformation (Box-Cox transformations). The following pictures shows the distribution before and after the transformation.

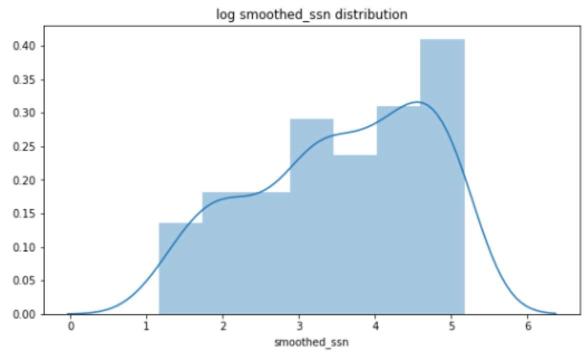












The output distributions are not normal; however, this will be enough to get good results.

We computed a series of rolling statistical measures over windows of different lengths. For each of time series mentioned above is computed:

- The mean and standard deviation over the last 1, 5, 10 and 48 hours.
- The slope and the intercept values of the linear least-squares regression model trained on the last 48 hours of data.
- The mean and standard deviation for both the real and imaginary parts of the Fast Fourier Transform (FFT) over the last 72 hours. Then, we compute the mean, standard deviation and the 10%, 50% and 90% percentiles of the Power Spectrum (PSD).
- The change rate and CID over the last 48 hours of the time series. These are used to measure the complexity of the time series, the formula of each operation is describe below.

$$CID = \sqrt{\frac{1}{n}\sum_{i=1}^{n-1}\left(x_i - x_{i-1}\right)^2}$$

$$mean\ change = \frac{1}{n}\sum_{i=1}^{n-1}\frac{x_i - x_{i-1}}{x_i}$$

- The length of the longest consecutive subsequence of the time series that lies within 1.96 standard deviations away from the mean.

After computing all solar wind features, we added the sunspots and satellite position data and apply column wise the following operations:

- Replace NaN values with the median of the distribution
- Normalize using Z-score normalization

Finally, we are left with a total of 186 features but we will reduce this number later!

Modeling

Our solution is an ensemble of 3 models: 1 Gradient Boosting Machine (using the LGBM implementation) and 2 Feed-Forward Neural Nets (NN) with dropout and batch normalization. In the case of the LGBM we train 2 models, one for each horizon (t and t + 1 hour). For the feed-forward NNs, we train only one model.

We computed a lot of features but most of them were redundant. To reduce the correlation in input space, we:

- Trained the model with all features
- Computed the feature importance
- Trained the model using only the most important ones

This approach reduced overfit, improved validation score and model complexity.

Copy and paste the 3 most impactful parts of your code and explain what each does and how it helped your model.

Permutation Importance

```
def permutation_importance(model, data,
                    features,
                    target,
                    score_func,
                    times: int = 1):

  def _score(data):
    _, dl = create_dl(data, features=features)
    output = predict_dl(model, dl)
    prediction = output['prediction'].numpy()
    error = score_func(data[target], prediction)
    return error

  base_score = _score(data)
  fi = []

  for feature in features:
    permuted_data = data.copy()
    permuted_data[feature] = np.random.permutation(permuted_data[feature])
    feature_score = _score(permuted_data)
    feature_importance = {'feature': feature,
                   'score': feature_score,
                   'importance': feature_score-base_score,
                   }
    fi.append(feature_importance)
  fi = pd.DataFrame(fi)
  fi.sort_values(by='importance', inplace=True, ascending=False)
  fi.reset_index(drop=True, inplace=True)
  return fi
```

We computed a lot of redundant features which leads to overfitting and more complex models. We implemented permutation importance to determine which features affects the prediction the most, and only used the most important ones. We used this technique to filter the features for NN based model. In the case of the Gradient Boosting Model, we used the built-in LGBM feature importance implementation.

The outline performance of this algorithm can be described in the following steps:

- Compute the reference score of the model using the raw data
- For each feature **j** in the dataset
    - Randomly shuffle the column **j** to generated a corrupted version of the data
    - Compute the RMSE on the corrupted data
    - Compute the importance of the feature defined as:

$$importance = score_{corrupted\ feature} - score_{base}$$

Fourier Features

```
def compute_fourier_stats(values):
    # computes the rfft
    rfft = np.fft.rfft(values)
    # power spectrum
    psd = np.real(rfft * np.conj(rfft)) / (len(values)**2)
    # real part
    real_rfft = np.real(rfft) / len(values)
    # imag part
    imag_rfft = np.imag(rfft) / len(values)
    # compute the stats
    stats = {'rfft_real_mean': real_rfft.mean(),
             'rfft_real_std': real_rfft.std(),
             'power_spectrum_mean': psd.mean(),
             'power_spectrum_q0.1': np.quantile(psd, 0.1),
             'power_spectrum_q0.5': np.quantile(psd, 0.5),
             'power_spectrum_q0.9': np.quantile(psd, 0.9),
             'power_spectrum_std': psd.std(),
             'rfft_imag_mean': imag_rfft.mean(),
             'rfft_imag_std': imag_rfft.std()}
    return stats
```

This function computes all Fourier transform statistics of the time series. We compute all of them for each of the solar wind time series (more information in the solution description section). These features improved significantly our performance in both local and leaderboard score.

Ensemble

```
def ensemble(data: pd.DataFrame,
        prediction: List[str] = default.yhat):
  """
  A function to ensemble preds by grouping by ['period', 'timedelta']
  # Parameters
  data: `pd.DataFrame`
    A dataframe containing the predictions
  prediction: `List[str]`, optional (defulat=['yhat_t0', 'yhat_t1'])
    the name of the prediction columns
  """
  ensemble_pred = data.groupby(['period', 'timedelta'])
  ensemble_pred = ensemble_pred[prediction].mean()
  ensemble_pred.reset_index(inplace=True)
  return ensemble_pred
```

We used ensembling for mixing the 2 feed-forward NNs and LGBM model (combining different models improve generalization most of the time). The ensembled model outperformed by 4% the local score (with respect to the single model).

Please provide the machine specs and time you used to run your model.

	@camaron_ai	@NataliaLaDelMar
CPU (model)	Intel i7-8550U	Intel i7 4 cores
GPU (model or N/A)	N/A	N/A
Memory (GB)	16	16
OS	Windows 10	Mac OS
Train duration	About 2h	About 2h
Inference duration	6-7h	6-7h

What are some other things you tried that didn't necessarily make it into the final workflow (quick overview)?

- We tried to reduce the size of the dataset by first aggregating the data at the hourly basis as preprocessing before computing features. We then realized that the aggregation step was removing some relevant information to solve the problem. We ended up using the raw time series for features computation. However, this approach takes longer time to run.

- Most of the features we computed were correlated. We tried dropping high correlated features before training, but we ended up dropping important features that could be important for modeling. Then, we decided to train the model first with all features and then drop some of them based on individual model's feature importance.

- We used PCA to capture important modes in the signal. However, this increased the number of features and the processing time significantly. More importantly, it did not improve the validation score.

- For feature generation, we tried a lot of statistical measures as:

    - Kurtosis
    - Skewness
    - Minimum Value
    - Maximum Value
    - Time since the last Peak
    - Time since the minimum value occurred
    - Number of Peaks
    - Fourier Coefficients
    - The number of consecutives values that are lower than the mean of the time series
    - The number of consecutives values that are higher than the mean of the time series
    - Percentile Values of the distribution

Did you use any tools for data preparation or exploratory data analysis that aren't listed in your code *submission?*

No, we did not.

How did you evaluate performance of the model other than the provided metric, if at all?

There are 3 periods on the training set. We focused on improving the RMSE on the “train_a” period which we considered was the hardest one to predict (since this period occurred in the peak of the solar cycle)

Anything we should watch out for or be aware of in using your model (e.g. code quirks, memory requirements, numerical stability issues, etc.)?

Certain parts of model training are stochastic especially in Neural Network Models (batch normalization, dropout, etc.), and model’s re-training might not result in the exactly the same model’s parameters.

Do you have any useful charts, graphs, or visualizations from the process?

No, we do not

If you were to continue working on this problem for the next year, what methods or techniques might you try in order to build on your work so far? Are there other fields or features you felt would have been very helpful to have?

We improved our local score before the competition ended. However, we did not get the chance to submit our last submission after teaming up. Thus, if we were to continue working on this problem we would:

- Add an input batch normalization layer in the NN’s architecture
- Remove even more redundant features using the model’s feature importance
- Add more models with different architectures and hyper parameters
- Try model stacking. We did not use it due to inference time competition constrains (less than 8 hours for the test set)