In EECS 16A, you saw the OMP algorithm as a method to find a sparse set of circularly-shifted vectors whose linear combination constituted a received signal. The motivation was in the context of decoding messages sent from satellites. It turns out that that this idea of finding a sparse-linear combination can also help us in Machine Learning. In particular it helps us with the problem of outlier removal.

1 Recap of OMP

Before we see how OMP is used in Machine Learning, let's do a brief recap of the algorithm by returning to the context in which EECS 16A covered it. Suppose we have m satellites, each periodically transmitting a unique code $\mathrm{s_i} \in \mathbb{R}^n$. Satellites encode messages by multiplying the code they transmit by some scalar $\alpha_i \in \mathbb{R}$. At any given moment, only k satellites are transmitting, so at the receiver, we measure the signal

$$\boldsymbol{y} = \sum_{i \in K} \alpha_i \boldsymbol{s_i}^{(\tau_i)}$$

where τ_i is a circular shift induced by the fact that each satellite is some distance away from the receiver and K is the set of all transmitting satellites. We need to find out which satellites are transmitting as well as the corresponding messages. One natural way to proceed is iteratively. We can first "predict" which satellites are transmitting by finding the code $\boldsymbol{s_i}$ and shift τ_k such that $\boldsymbol{s_i}^{(\tau_k)}$ has the largest absolute cross-correlation than any other $\boldsymbol{s_j}^{(\tau_p)}$ ($i, j \leq m$ and $k, p \leq n$). Then, we perform OLS to find the coefficient α_i of $\boldsymbol{s_i}^{(\tau_k)}$, giving us how much $\boldsymbol{s_i}^{\tau_k}$ "explains" our received signal. Finally, we compute the residual $\boldsymbol{e} = \boldsymbol{y} - \alpha_i \boldsymbol{s_i}^{(\tau_k)}$, which tells us how much of our signal we still need to explain, and repeat the process on the residual $k - 1$ times (or until our residual is small). Each time we repeat, we re-compute the α_i since finding more of the satellites will give us a different estimate of α_i each time. This process is summarized by algorithm 1.

2 OMP for Outlier Removal in Linear Regression

On its face, the setup for OMP seems entirely different from our Linear Regression setup. In Linear Regression, we have n targets $y_i = \boldsymbol{w}^T \boldsymbol{x_i} + \epsilon_i$ which are the sum of features $\boldsymbol{x_i} \in \mathbb{R}^m$ weighted by $\boldsymbol{w}$ with some noise ϵ_i. In matrix form, this is

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} \boldsymbol{x_1}^T \\ \boldsymbol{x_2}^T \\ \vdots \\ \boldsymbol{x_n}^T \end{bmatrix} \boldsymbol{w} + \boldsymbol{\epsilon}.$$

It might not be entirely obvious at first, but we can interpret this exactly as an OMP problem by properly deciding what to call our "message" and what to call

Algorithm 1: OMP Algorithm from 16A

Input: Codes $\boldsymbol{s_i}$, Received signal $\boldsymbol{y}$, Sparsity k, Residual Threshold ϵ
Output: The indices of the used codes F, the sent messages $\boldsymbol{x}$
$\boldsymbol{e} = \boldsymbol{y}$, j = 1, A = [], F = $\emptyset$;
while $j \leq k$ *and* $||\boldsymbol{e}|| \geq \epsilon$ **do**
    i, t = FindMaxCrossCorrelation();
    $F = F \bigcup \{i\}$;
    $A = [A|s_i^{(t)}]$;
    $\boldsymbol{x} = (A^T A)^{-1} A^T \boldsymbol{y}$;
    $\boldsymbol{e} = \boldsymbol{y} - A\boldsymbol{x}$;
    j = j + 1;
end
return $F, \boldsymbol{x}$

our "codes". First, let's express the OMP problem in matrix form.

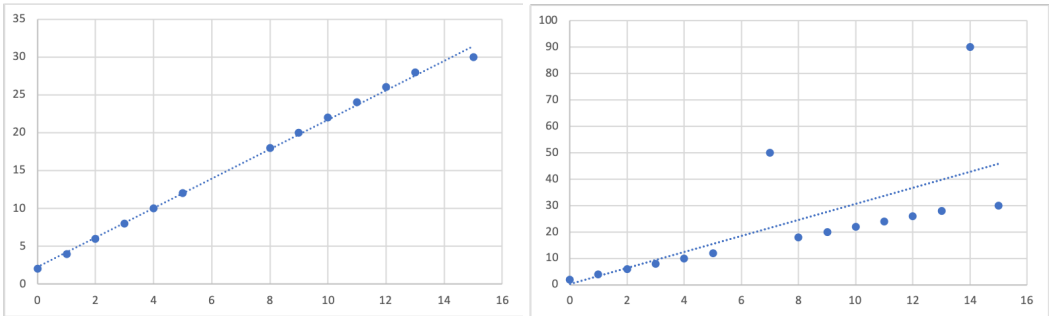
$$\boldsymbol{y} = \begin{bmatrix} \boldsymbol{s_1}^{(\tau_1)} & \boldsymbol{s_2}^{(\tau_2)} & \cdots & \boldsymbol{s_m}^{(\tau_m)} \end{bmatrix} \boldsymbol{\alpha}$$

If we ignore the noise in the Machine Learning context and the shifts in the OMP context, clearly our received signal $\boldsymbol{y}$ can be thought of as our targets, and the sparse message $\boldsymbol{\alpha}$ is the weights of the model. The codes $\boldsymbol{s_i}$ are simply the columns of the data matrix (i.e the features of each data point). Notice that we can ignore the shifts τ_i because there is nothing in Linear Regression which suggests a "delay", and the noise in the machine learning context translates to the OMP context naturally because there is no guarantee $\boldsymbol{y}$ is received perfectly.

2.1 Outliers

How does this interpretation help us with outlier removal exactly? An outlier is a point which lies very far from the true linear model. Mathematically, it means for the target y_i, ϵ_i is very large, meaning it is either corrupted by a lot of noise or was generated with a different process than the linear model, so it can't be explained by our linear model.

Because these points can't be well explained by a linear model, if we were to run OLS on data containing outliers, the outliers would skew the model away from the actual fit because OLS will try and balance minimizing the residual to the outliers against minimizing the residual from all of the uncorrupted data. Notice how in fig. 1a where there is only a small amount of noise, the learned model reflects the general trend of the data (all of the residuals are small). However, in fig. 1b, the two outliers increase the slope of the line (in an attempt to make the residuals to those two points smaller), decreasing its ability to capture the correct trend.



(a) Model learned without outliers (b) Model learned with outliers

Figure 1

2.2 Interpreting OMP as an Outlier Removal Problem

How does OMP help us solve the problem of outliers? Well if we knew which points were outliers, we could remove them from our training set and train with the remaining data. Recall that

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} \boldsymbol{x_1}^T \\ \boldsymbol{x_2}^T \\ \vdots \\ \boldsymbol{x_n}^T \end{bmatrix} \boldsymbol{w} + \boldsymbol{\epsilon}.$$

If we break up $\boldsymbol{\epsilon}$, we see,

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} \boldsymbol{x_1}^T \\ \boldsymbol{x_2}^T \\ \vdots \\ \boldsymbol{x_n}^T \end{bmatrix} \boldsymbol{w} + \sum \epsilon_i \boldsymbol{e_i} = \begin{bmatrix} X & I \end{bmatrix} \begin{bmatrix} \boldsymbol{w} \\ \boldsymbol{\epsilon} \end{bmatrix}$$

where $\boldsymbol{e_i}$ are the standard basis vectors. What this tells us is that outliers are essentially large perturbations along the standard basis in feature space. Since we only have a few outliers, $\boldsymbol{\epsilon}$ is sparse.

Thinking about this in the context of OMP, we already know that the features of our data matrix can explain our signal. We now need to find the standard basis vectors which also explain our signal. In the context of satellites, it is as if we have m satellites transmitting their codes normally but then k "adversarial" satellites very loudly transmit one of the standard basis vectors, effectively drowning out the other satellites. If we find the loud, "adversarial" satellites, then we can remove them and recover our true message again.

This interpretation naturally leads us to OMP initialized with $A = X$, proceeding as normal thereafter. We find the $\boldsymbol{e_i}$ which most explains our residual, indicating that the ith point is an outlier, augment A with $\boldsymbol{e_i}$, and continue. At the end of the

algorithm, we output the set F which tells us the indices of the outliers. Once we remove the outliers from the data set, we can proceed as normal with OLS.

Note that because there are no delays, we don't need to compute the circular cross-correlation. We only need to find the maximum, absolute dot product of each standard basis vector with the residual $\boldsymbol{r}$. In other words, we need to find

$$i = \underset{i}{\operatorname{argmax}} |\langle \boldsymbol{r}, \boldsymbol{e_i} \rangle| = \underset{i}{\operatorname{argmax}} |\boldsymbol{r}[i]|$$

where $r[i]$ is the residual for the ith datapoint.

Algorithm 2: OMP to Detect Outliers

Input: Data matrix X, Predictions $\boldsymbol{y}$, Sparsity k, Residual Threshold ϵ
Output: Set of data indices F where the outliers are
$\boldsymbol{r} = \boldsymbol{y} - X(X^TX)^{-1}X^T\boldsymbol{y}$, j = 1, A = X, F = $\emptyset$;
while $j \leq k$ *and* $||\boldsymbol{r}|| \geq \epsilon$ **do**
  i = $\operatorname{argmax}_i |\boldsymbol{r}[i]|$;
  $F = F \bigcup \{i\}$;
  $A = [A|\boldsymbol{e_i}]$;
  $\boldsymbol{x} = (A^TA)^{-1}A^T\boldsymbol{y}$;
  $\boldsymbol{r} = \boldsymbol{y} - A\boldsymbol{x}$;
  j = j + 1;
end
return F

It turns out that we do not have to initialize OMP with the data matrix X. Remember that OMP is selecting the features which best explain our data, so ideally, our actual features should be the ones that best explain the data, so OMP will choose them anyway if we run it on the matrix $[X|I]$. However, because OMP is greedily looking for the loudest features, it is possible that if the features are not scaled properly, then OMP might not choose some of our true features because they are too "quite". This is why it is very important that we engineer our features appropriately to make sure this doesn't happen.

2.3 Stopping Conditions

At this point, you might be asking yourself, "How do we know how many outliers are in the dataset?" or "How do we know when to stop running OMP?". This question has 3 natural answers for linear regression.

Choose k: If you know the number of outliers is k, then you only need to run OMP for k iterations. You might be able to find a good k by visualizing the dataset and seeing which points deviate significantly from a linear relationship.

Choose ϵ: You can also stop OMP when the residual gets "too small". The value that determines "too small" would be determined by looking at your validation error and finding the ϵ corresponding to the minimum MSE.

Trivial Case: If you only have 2 data points (either by running OMP for too many iterations or just because you have a small dataset), then you can always fit a line that perfectly predicts both points since a line is defined by 2 points. If we reached 2 points by running OMP, then we should stop iterating.