Iterative Approximate Cross Validation in High Dimensions

Sina Alsharif

School of Computer Science and Engineering, University of New South Wales

Thesis A (Term 2, 2023)

Section 1

Background

Background

To set out the notation used throughout this seminar, we can define the Empirical Risk Minimisation (ERM) framework to solve a supervised learning problem.

Background

To set out the notation used throughout this seminar, we can define the Empirical Risk Minimisation (ERM) framework to solve a supervised learning problem.

General Problem Setting

- Input space $\mathcal{X}$ and an output space $\mathcal{Y}$.
- Data is "generated" by a *true* distribution $P(\mathcal{X}, \mathcal{Y})$.
- Aim is to find a mapping $h : \mathcal{X} \to \mathcal{Y}$ (called a hypothesis).
- Denote all possible combinations of input and output space as $\mathcal{D} = \{(X, Y) \in (\mathcal{X}, \mathcal{Y})\}$.

Risk

To measure the error (or loss) we make on a data point, define a function $\ell(h; D_i)$,

$$\ell(h; D) = \sum_{i=1}^{n} \ell(h; D_i) \text{ where } D_i = (X_i, Y_i)$$

as the loss for the *observed* data $D = \{(X_i, Y_i)\}_{i=1}^{n}$. Examples of ℓ are 0-1 loss (for classification) and a squared error (for regression).

Risk

To measure the error (or loss) we make on a data point, define a function $\ell(h; D_i)$,

$$\ell(h; D) = \sum_{i=1}^{n} \ell(h; D_i) \text{ where } D_i = (X_i, Y_i)$$

as the loss for the *observed* data $D = \{(X_i, Y_i)\}_{i=1}^{n}$. Examples of ℓ are 0-1 loss (for classification) and a squared error (for regression).

Risk

Define the **risk** of a hypothesis for the complete space of inputs and outputs as,

$$R(h) = \mathbb{E}_{\mathcal{X},\mathcal{Y}}[\ell(h; \mathcal{D})]$$

The optimal hypothesis selected from a hypothesis space $\mathcal{H}$ is defined as,

$$h_{\mathcal{H}} = \arg\min_{h \in \mathcal{H}} R(h)$$

Empirical Risk

As we cannot measure **true** risk, we seek an approximation using the observed data D.

Empirical Risk

We define **empirical risk** of a hypothesis given data D as,

$$R_{\text{emp}}(h; D) = \ell(h; D)$$

The optimal hypothesis *given observed data* D is,

$$h_D = \arg\min_{h \in \mathcal{H}} R_{\text{emp}}(h; D)$$

where $\mathcal{H}$ is a hypothesis space which a *learning algorithm* picks a hypothesis from. We will describe the parameters which describe h as $\theta \in \mathbb{R}^p$, assuming the case of a Generalised Linear Model (GLM).

Issues with ERM

By the (weak) law of large numbers $R_{\mathsf{emp}}(h; D) \xrightarrow{P} R(h; \mathcal{D})$ as $n \to \infty$, so it is reasonable to assume that h_D converges to a minimiser of true risk.

[1] Refer to the bias-variance (or approximation-estimation) tradeoff.

Issues with ERM

By the (weak) law of large numbers $R_{\text{emp}}(h; D) \xrightarrow{P} R(h; \mathcal{D})$ as $n \to \infty$, so it is reasonable to assume that h_D converges to a minimiser of true risk. However, if we fit the explicit minimiser of empirical risk, we will not always find the minimiser of true risk [1]. We can restrict the learning algorithm's ability to fully minimise empirical risk through **regularisation**.

[1] Refer to the bias-variance (or approximation-estimation) tradeoff.

Issues with ERM

By the (weak) law of large numbers $R_{\text{emp}}(h; D) \xrightarrow{P} R(h; \mathcal{D})$ as $n \to \infty$, so it is reasonable to assume that h_D converges to a minimiser of true risk. However, if we fit the explicit minimiser of empirical risk, we will not always find the minimiser of true risk [1]. We can restrict the learning algorithm's ability to fully minimise empirical risk through **regularisation**.

Regularised Empirical Risk

To restrict the hypothesis space, we define the formulation of **regularised empirical risk** as,

$$R_{\text{reg}}(h; D) = R_{\text{emp}}(h; D) + \lambda\pi(\theta)$$

Here, $\pi : \mathbb{R}^p \to \mathbb{R}$ is a regulariser and $\lambda \in \mathbb{R}^+$ is a hyper-parameter to control the strength of regularisation. The solution becomes $(h_D)_\lambda = \arg\min_{h \in \mathcal{H}_\lambda} R_{\text{emp}}(h; D)$ where $\mathcal{H}_\lambda$ is a restricted hypothesis space.

[1] Refer to the bias-variance (or approximation-estimation) tradeoff.

Cross Validation

To avoid "overfitting" to the observed data (i.e blindly minimising empirical risk), we can attempt to define an approximation of true risk to measure the efficacy of a learned hypothesis.

[2] Arlot & Celisse (2008)

Cross Validation

To avoid "overfitting" to the observed data (i.e blindly minimising empirical risk), we can attempt to define an approximation of true risk to measure the efficacy of a learned hypothesis.

The most common way to estimate the true risk of a hypothesis is to run Cross Validation (CV) for a hypothesis. This is where we break the observed data into small subsets to run multiple "validation" experiments (training the data on a subset and testing on an unseen subset).

[2] Arlot & Celisse (2008)

Cross Validation

To avoid "overfitting" to the observed data (i.e blindly minimising empirical risk), we can attempt to define an approximation of true risk to measure the efficacy of a learned hypothesis.

The most common way to estimate the true risk of a hypothesis is to run Cross Validation (CV) for a hypothesis. This is where we break the observed data into small subsets to run multiple "validation" experiments (training the data on a subset and testing on an unseen subset).

One of the most effective methods for risk approximation is Leave One Out Cross Validation (LOOCV) [2]. This method is computationally expensive as we repeat the learning task n times (where n is the size of the observed data).

[2] Arlot & Celisse (2008)

Section 2

Literature Review

Approximate CV Methods

To reduce the computational cost of CV (specifically LOOCV), we turn to Approximate Cross Validation (ACV), which attempts to approximate (rather than solve) individual CV experiments.

LOOCV

The definition of leave-one-out (LOO) regularised empirical risk is,

$$R_{\text{reg}}(\theta; D_{-j}) = \sum_{i=1, i\neq j}^{n} \ell(\theta; D_i) + \lambda\pi(\theta)$$

where we leave out a point with index j for this experiment.

Methods for ACV

There are three main methods for ACV we will discuss.

- Newton Step ("NS")
- Infinitesimal Jackknife ("IJ")
- Iterative Approximate Cross Validation ("IACV")

both NS and IJ are existing methods, and IACV is a new proposed method which we aim to adapt and extend.

Methods for ACV

There are three main methods for ACV we will discuss.

- Newton Step ("NS")
- Infinitesimal Jackknife ("IJ")
- Iterative Approximate Cross Validation ("IACV")

both NS and IJ are existing methods, and IACV is a new proposed method which we aim to adapt and extend.

We will brush over the theory for NS and IJ in the following section, taking a closer look at the derivation of IACV and its proposed improvements on the former methods.

It is important to note that these methods are only used when the learning task is solved through an iterative method, where the ACV steps are tacked on to the end of an iteration.

Newton Step

We can redefine the definition for (regularised) empirical risk for a LOOCV experiment excluding a point with index j,

$$R_{\text{reg}}(\theta; D_{-j}) = \sum_{i=1}^{n} \ell(\theta; D_i) - \ell(\theta; D_j) + \lambda\pi(\theta)$$

Newton Step

We can redefine the definition for (regularised) empirical risk for a LOOCV experiment excluding a point with index j,

$$R_{\text{reg}}(\theta; D_{-j}) = \sum_{i=1}^{n} \ell(\theta; D_i) - \ell(\theta; D_j) + \lambda\pi(\theta)$$

The Jacobian of this form is,

$$\nabla_\theta R_{\text{reg}}(\theta; D_{-j}) = \sum_{i=1}^{n} \nabla_\theta \ell(\theta; D_i) - \nabla_\theta \ell(\theta; D_j) + \lambda \nabla_\theta \pi(\theta)$$

Therefore the Hessian becomes,

$$\nabla^2_\theta R_{\text{reg}}(\theta; D_{-j}) = H(\theta; D) - \nabla^2_\theta \ell(\theta, D_j)$$

$$\text{where} \quad H(\theta; D) = \nabla^2_\theta \left(\sum_{i=1}^{n} \ell(\theta; D_i) \right) + \lambda \nabla^2_\theta \pi(\theta)$$

Therefore the Hessian becomes,

$$\nabla^2_\theta R_{\text{reg}}(\theta; D_{-j}) = H(\theta; D) - \nabla^2_\theta \ell(\theta, D_j)$$

$$\text{where} \quad H(\theta; D) = \nabla^2_\theta \left(\sum_{i=1}^{n} \ell(\theta; D_i) \right) + \lambda \nabla^2_\theta \pi(\theta)$$

Note that we assume $\ell(\cdot)$ and $\pi(\cdot)$ are both continous and twice-differentiable functions.

Therefore the Hessian becomes,

$$\nabla_\theta^2 R_{\text{reg}}(\theta; D_{-j}) = H(\theta; D) - \nabla_\theta^2 \ell(\theta, D_j)$$

$$\text{where} \quad H(\theta; D) = \nabla_\theta^2 \left(\sum_{i=1}^{n} \ell(\theta; D_i) \right) + \lambda \nabla_\theta^2 \pi(\theta)$$

Note that we assume $\ell(\cdot)$ and $\pi(\cdot)$ are both continous and twice-differentiable functions.

Now we can apply Newton's method for optimisation to take a "step" towards the LOOCV iterate $\hat{\theta}_{-j}$ by starting at the learned parameter $\hat{\theta}$.

We define the approximation of a LOOCV iterate as $\tilde{\theta}_{-j}$, where

$$\begin{aligned}\tilde{\theta}_{-j} &= \hat{\theta} - \Big(H(\hat{\theta}; D) - \nabla^2_\theta \ell(\hat{\theta}; D_j)\Big)^{-1} \Big(\nabla_\theta R_{\text{reg}}(\hat{\theta}; D) - \nabla_\theta R_{\text{reg}}(\hat{\theta}; D_j)\Big) \\ &= \hat{\theta} + \Big(H(\hat{\theta}; D) - \nabla^2_\theta \ell(\hat{\theta}; D_j)\Big)^{-1} \nabla_\theta R_{\text{reg}}(\hat{\theta}; D_j)\end{aligned}$$

the second line follows by the definition of $\hat{\theta}$. Note here, we also assume that the (modified) Hessian $\Big(H(\hat{\theta}; D) - \nabla^2_\theta \ell(\hat{\theta}; D_j)\Big)$ is invertible.

We define the approximation of a LOOCV iterate as $\tilde{\theta}_{-j}$, where

$$\begin{aligned}\tilde{\theta}_{-j} &= \hat{\theta} - \left(H(\hat{\theta}; D) - \nabla^2_\theta \ell(\hat{\theta}; D_j)\right)^{-1} \left(\nabla_\theta R_{\text{reg}}(\hat{\theta}; D) - \nabla_\theta R_{\text{reg}}(\hat{\theta}; D_j)\right) \\ &= \hat{\theta} + \left(H(\hat{\theta}; D) - \nabla^2_\theta \ell(\hat{\theta}; D_j)\right)^{-1} \nabla_\theta R_{\text{reg}}(\hat{\theta}; D_j)\end{aligned}$$

the second line follows by the definition of $\hat{\theta}$. Note here, we also assume that the (modified) Hessian $\left(H(\hat{\theta}; D) - \nabla^2_\theta \ell(\hat{\theta}; D_j)\right)$ is invertible.

For discussion, the standard notation we'll use for the NS method is,

$$\tilde{\theta}^{-i}_{\text{NS}} = \hat{\theta} + \left(H(\hat{\theta}; D) - \nabla^2_\theta \ell(\hat{\theta}; D_i)\right)^{-1} \nabla_\theta R_{\text{reg}}(\hat{\theta}; D_i)$$

Infinitesimal Jackknife

An alternative method for ACV is the infinitesimal Jackknife (IJ). The complete derivation has been omitted for the sake of brevity, however the general idea is to use a *weighted* loss (i.e $w_i \ell(\theta; D_i)$ where $w_i \in \mathbb{R}^+$ and we have a $w = \{w_i\}_{i=1}^n$) and perform a first-order Taylor expansion around the weights to approximate LOOCV.

The final form derived for this case is

$$\tilde{\theta}_{\mathsf{IJ}}^{-i} = \hat{\theta} + (H(\hat{\theta}; D))^{-1} \nabla_\theta R_{\mathsf{reg}}(\hat{\theta}; D_i)$$

where we again make the same assumptions as in NS (loss and regularisation are continuously twice-differentiable, H is invertible).

Infinitesimal Jackknife

An alternative method for ACV is the infinitesimal Jackknife (IJ). The complete derivation has been omitted for the sake of brevity, however the general idea is to use a *weighted* loss (i.e $w_i \ell(\theta; D_i)$ where $w_i \in \mathbb{R}^+$ and we have a $w = \{w_i\}_{i=1}^n$) and perform a first-order Taylor expansion around the weights to approximate LOOCV.

The final form derived for this case is

$$\tilde{\theta}_{\mathsf{IJ}}^{-i} = \hat{\theta} + (H(\hat{\theta}; D))^{-1} \nabla_\theta R_{\mathsf{reg}}(\hat{\theta}; D_i)$$

where we again make the same assumptions as in NS (loss and regularisation are continuously twice-differentiable, H is invertible). This method has a computational advantage over NS, as we only need to calculate and invert $H(\hat{\theta}; D)$ once, rather than n times.

Iterative Approximate Cross Validation

A recently proposed method for ACV is Iterative Approximate Cross Validation (IACV) and aims to improve on existing methods by relaxing assumptions and providing accurate approximation before convergence of the main iterative method.

We again solve the main learning task through an iterative method, where the updates are (for GD and SGD)

$$\hat{\theta}^{(k)} = \hat{\theta}^{(k-1)} - \alpha_k \nabla_\theta R_{\text{reg}}(\hat{\theta}^{(k-1)}; D_{S_k})$$

where $S_k \subseteq [n]$ is a subset of indices and α_k is a learning rate taken for an iteration k. For classic GD, $S_k \equiv [n]$ and can be variable for SGD.

Iterative Approximate Cross Validation

A recently proposed method for ACV is Iterative Approximate Cross Validation (IACV) and aims to improve on existing methods by relaxing assumptions and providing accurate approximation before convergence of the main iterative method.

We again solve the main learning task through an iterative method, where the updates are (for GD and SGD)

$$\hat{\theta}^{(k)} = \hat{\theta}^{(k-1)} - \alpha_k \nabla_\theta R_{\text{reg}}(\hat{\theta}^{(k-1)}; D_{S_k})$$

where $S_k \subseteq [n]$ is a subset of indices and α_k is a learning rate taken for an iteration k. For classic GD, $S_k \equiv [n]$ and can be variable for SGD.

The explicit optimisation step LOOCV iterate excluding a point i is defined as,

$$\hat{\theta}_{-i}^{(k)} = \hat{\theta}_{-i}^{(k-1)} - \alpha_k \nabla_\theta R_{\text{reg}}(\hat{\theta}_{-i}^{(k-1)}; D_{S_k \setminus i})$$

this step is what we aim to approximate.

The main computational burden in the LOOCV optimisation step is calculating the Jacobian $\nabla_\theta R_{\text{reg}}(\hat{\theta}_{-i}^{(k-1)}; D_{S_t \setminus i})$ for n points.

The main computational burden in the LOOCV optimisation step is calculating the Jacobian $\nabla_\theta R_{\text{reg}}(\hat{\theta}_{-i}^{(k-1)}; D_{S_t \setminus i})$ for n points. If we use a second-order expansion of the Jacobian for $\hat{\theta}_{-i}^{(k-1)}$ centered around the estimate $\hat{\theta}^{(k-1)}$, we can approximate the Jacobian for this step.

The main computational burden in the LOOCV optimisation step is calculating the Jacobian $\nabla_\theta R_{\text{reg}}(\hat{\theta}_{-i}^{(k-1)}; D_{S_t \setminus i})$ for n points. If we use a second-order expansion of the Jacobian for $\hat{\theta}_{-i}^{(k-1)}$ centered around the estimate $\hat{\theta}^{(k-1)}$, we can approximate the Jacobian for this step. Here,

$$\nabla_\theta R_{\text{reg}}(\hat{\theta}_{-i}^{(k-1)}; D_{S_k \setminus i}) \approx \nabla_\theta R_{\text{reg}}(\hat{\theta}^{(k-1)}; D_{S_k \setminus i}) + \nabla_\theta^2 R_{\text{reg}}(\hat{\theta}^{(k-1)}; D_{S_k \setminus i}) \left(\tilde{\theta}_{-i}^{(k-1)} - \hat{\theta}^{(k-1)} \right)$$

is the estimate for the Jacobian.

The main computational burden in the LOOCV optimisation step is calculating the Jacobian $\nabla_\theta R_{\text{reg}}(\hat{\theta}_{-i}^{(k-1)}; D_{S_t \setminus i})$ for n points. If we use a second-order expansion of the Jacobian for $\hat{\theta}_{-i}^{(k-1)}$ centered around the estimate $\hat{\theta}^{(k-1)}$, we can approximate the Jacobian for this step. Here,

$$\nabla_\theta R_{\text{reg}}(\hat{\theta}_{-i}^{(k-1)}; D_{S_k \setminus i}) \approx \nabla_\theta R_{\text{reg}}(\hat{\theta}^{(k-1)}; D_{S_k \setminus i}) + \nabla_\theta^2 R_{\text{reg}}(\hat{\theta}^{(k-1)}; D_{S_k \setminus i}) \left(\tilde{\theta}_{-i}^{(k-1)} - \hat{\theta}^{(k-1)} \right)$$

is the estimate for the Jacobian.

Therefore, the IACV updates for GD and SGD become,

$$\tilde{\theta}_{-i}^{(k)} = \tilde{\theta}_{-i}^{(k-1)} - \alpha_k \left(\nabla_\theta R_{\text{reg}}(\hat{\theta}^{(k-1)}; D_{S_k \setminus i}) + \nabla_\theta^2 R_{\text{reg}}(\hat{\theta}^{(k-1)}; D_{S_k \setminus i}) \left(\tilde{\theta}_{-i}^{(k-1)} - \hat{\theta}^{(k-1)} \right) \right)$$

The main difference (from NS and IJ) is that we can define an ACV update rule for proximal gradient descent.

The main difference (from NS and IJ) is that we can define an ACV update rule for proximal gradient descent. If we define a general update rule for LOOCV proximal gradient descent as,

$$\hat{\theta}_{-i}^{(k)} = \arg\min_{z} \left\{ \frac{1}{2\alpha_k} \|z - \theta'_{-i}\|_2^2 + \lambda\pi(z) \right\}$$
$$\text{where } \theta'_{-i} = \hat{\theta}_{-i}^{(k-1)} - \alpha_k \nabla_\theta \ell(\hat{\theta}_{-i}^{(k-1)}; D_{S_k \setminus i})$$

using similar logic as in GD/SGD on the differentiable part of the regularised risk, we get IACV updates of,

The main difference (from NS and IJ) is that we can define an ACV update rule for proximal gradient descent. If we define a general update rule for LOOCV proximal gradient descent as,

$$\hat{\theta}_{-i}^{(k)} = \arg\min_{z} \left\{ \frac{1}{2\alpha_k} \|z - \theta'_{-i}\|_2^2 + \lambda\pi(z) \right\}$$
$$\text{where } \theta'_{-i} = \hat{\theta}_{-i}^{(k-1)} - \alpha_k \nabla_\theta \ell(\hat{\theta}_{-i}^{(k-1)}; D_{S_k \setminus i})$$

using similar logic as in GD/SGD on the differentiable part of the regularised risk, we get IACV updates of,

$$\tilde{\theta}_{-i}^{(k)} = \arg\min_{z} \left\{ \frac{1}{2\alpha_k} \|z - \theta'_{-i}\|_2^2 + \lambda\pi(z) \right\}$$
$$\text{where } \theta'_{-i} = \tilde{\theta}_{-i}^{(k-1)} - \alpha_k \left(\nabla_\theta \ell(\hat{\theta}^{(k-1)}; D_{S_k \setminus i}) + \nabla_\theta^2 \ell(\hat{\theta}^{(k-1)}; D_{S_k \setminus i}) \left(\tilde{\theta}_{-i}^{(k-1)} - \hat{\theta}^{(k-1)} \right) \right)$$

It may seem counter-intuitive that we swap a simple Jacobian for a Jacobian *and* a Hessian in the approximation step.

[3] Luo & Barber (2023)

It may seem counter-intuitive that we swap a simple Jacobian for a Jacobian *and* a Hessian in the approximation step. The time complexities for the operations are as follows,

	IACV	Exact LOOCV
GD	$n(A_p + B_p) + np^2$	$n^2 A_p + np$
SGD	$K(A_p + B_p) + np^2$	$nKA_p + np$
ProxGD	$n(A_p + B_p + D_p) + np^2$	$n^2 A_p + nD_p + np$

where A_p is one evaluation of the Jacobian, B_p is one evaluation of the Hessian, D_p is one evaluation of the proximal operator and K is the size of the subset used for SGD [3].

[3] Luo & Barber (2023)

It may seem counter-intuitive that we swap a simple Jacobian for a Jacobian *and* a Hessian in the approximation step. The time complexities for the operations are as follows,

	IACV	Exact LOOCV
GD	$n(A_p + B_p) + np^2$	$n^2 A_p + np$
SGD	$K(A_p + B_p) + np^2$	$nKA_p + np$
ProxGD	$n(A_p + B_p + D_p) + np^2$	$n^2 A_p + nD_p + np$

where A_p is one evaluation of the Jacobian, B_p is one evaluation of the Hessian, D_p is one evaluation of the proximal operator and K is the size of the subset used for SGD [3]. The time complexities here however, are not representative of empirics as the Jacobian and Hessian for IACV can be easily vectorised since we hold the argument $\hat{\theta}^{(k-1)}$ fixed.

[3] Luo & Barber (2023)

It may seem counter-intuitive that we swap a simple Jacobian for a Jacobian *and* a Hessian in the approximation step. The time complexities for the operations are as follows,

	IACV	Exact LOOCV
GD	$n(A_p + B_p) + np^2$	$n^2 A_p + np$
SGD	$K(A_p + B_p) + np^2$	$nKA_p + np$
ProxGD	$n(A_p + B_p + D_p) + np^2$	$n^2 A_p + nD_p + np$

where A_p is one evaluation of the Jacobian, B_p is one evaluation of the Hessian, D_p is one evaluation of the proximal operator and K is the size of the subset used for SGD [3]. The time complexities here however, are not representative of empirics as the Jacobian and Hessian for IACV can be easily vectorised since we hold the argument $\hat{\theta}^{(k-1)}$ fixed.

In terms of space IACV uses $O(np + p^2)$ space, where as exact LOOCV uses $O(np)$ space. The orders of space are the similar when $p \ll n$, however can be an issue in higher dimensional problems.

[3] Luo & Barber (2023)

Problems In High Dimensions

The aforementioned ACV methods face problems in higher dimensions. For IJ and NS, the main problems are [4]

- No form for ℓ_1 methods used in high dimensions
- Time complexity breakdown (especially for Hessian inversion)
- A breakdown of accuracy

We assume a similar theme of time (and memory) complexity issues with IACV in higher dimensions. A loss of accuracy however, is unclear and needs further investigation.

[4] Stephenson & Broderick (2020)

Problems In High Dimensions

The aforementioned ACV methods face problems in higher dimensions. For IJ and NS, the main problems are [4]

- No form for ℓ_1 methods used in high dimensions
- Time complexity breakdown (especially for Hessian inversion)
- A breakdown of accuracy

We assume a similar theme of time (and memory) complexity issues with IACV in higher dimensions. A loss of accuracy however, is unclear and needs further investigation.

Current error bounds for IACV do assume $n \leq p$, though preliminary empirics show that accuracy *may* not suffer greatly.

[4] Stephenson & Broderick (2020)

Existing Solutions for High Dimensional Problems

There are varying solutions for both NS and IJ for higher dimensional problems. Some solutions turn to smoothing ℓ_1 to ensure differentiability and randomised solvers to solve for the inverse of the Hessian.

Existing Solutions for High Dimensional Problems

There are varying solutions for both NS and IJ for higher dimensional problems. Some solutions turn to smoothing ℓ_1 to ensure differentiability and randomised solvers to solve for the inverse of the Hessian. However, the main solution to look to is one outlined in (Stephenson & Broderick (2020)).

Existing Solutions for High Dimensional Problems

There are varying solutions for both NS and IJ for higher dimensional problems. Some solutions turn to smoothing ℓ_1 to ensure differentiability and randomised solvers to solve for the inverse of the Hessian. However, the main solution to look to is one outlined in (Stephenson & Broderick (2020)).

The main idea of this solution is to work in the "effective dimension" or support of the data.

Existing Solutions for High Dimensional Problems

There are varying solutions for both NS and IJ for higher dimensional problems. Some solutions turn to smoothing ℓ_1 to ensure differentiability and randomised solvers to solve for the inverse of the Hessian. However, the main solution to look to is one outlined in (Stephenson & Broderick (2020)).

The main idea of this solution is to work in the "effective dimension" or support of the data. This reduces both computational complexity and possible issues of inversion. We first run an ℓ_1 learning task, and run ACV on the support at each iteration.

Existing Solutions for High Dimensional Problems

There are varying solutions for both NS and IJ for higher dimensional problems. Some solutions turn to smoothing ℓ_1 to ensure differentiability and randomised solvers to solve for the inverse of the Hessian. However, the main solution to look to is one outlined in (Stephenson & Broderick (2020)).

The main idea of this solution is to work in the "effective dimension" or support of the data. This reduces both computational complexity and possible issues of inversion. We first run an ℓ_1 learning task, and run ACV on the support at each iteration. If we define the estimated support of our data as $\hat{S}$, the updates for NS become,

$$\left[\tilde{\theta}_{\text{NS}}^{-i}\right]_j = \begin{cases} 0 & \text{when } \hat{\theta}_j = 0 \\ \hat{\theta}_j + \left[\left(H_{\hat{S}}(\hat{\theta}_{\hat{S}}; D) - \nabla^2_{\theta}\ell_{\hat{S}}(\hat{\theta}_{\hat{S}}; D_{-i})\right)^{-1} \nabla_{\theta}\ell_{\hat{S}}(\hat{\theta}_{\hat{S}}; D_i)\right]_j & \text{otherwise} \end{cases}$$

where we only evaluate the terms in the support.

Similarly, the "sparse ACV" updates for IJ becomes,

$$\left[\tilde{\theta}_{\text{IJ}}^{-i}\right]_j = \begin{cases} 0 & \text{when } \hat{\theta}_j = 0 \\ \hat{\theta}_j + \left[\left(H_{\hat{S}}(\hat{\theta}_{\hat{S}}; D)\right)^{-1} \nabla_\theta \ell_{\hat{S}}(\hat{\theta}_{\hat{S}}; D_i)\right]_j & \text{otherwise} \end{cases}$$

Similarly, the "sparse ACV" updates for IJ becomes,

$$\left[\tilde{\theta}_{\text{IJ}}^{-i}\right]_j = \begin{cases} 0 & \text{when } \hat{\theta}_j = 0 \\ \hat{\theta}_j + \left[\left(H_{\hat{S}}(\hat{\theta}_{\hat{S}}; D)\right)^{-1} \nabla_\theta \ell_{\hat{S}}(\hat{\theta}_{\hat{S}}; D_i)\right]_j & \text{otherwise} \end{cases}$$

Alongside computational benefits, reducing the dimension of the data used for ACV from n to $|\hat{S}|$ also allows for more accurate approximation in higher dimensions (given we make a few assumptions on the data).

Similarly, the "sparse ACV" updates for IJ becomes,

$$\left[\tilde{\theta}_{\text{IJ}}^{-i}\right]_j = \begin{cases} 0 & \text{when } \hat{\theta}_j = 0 \\ \hat{\theta}_j + \left[\left(H_{\hat{S}}(\hat{\theta}_{\hat{S}}; D)\right)^{-1} \nabla_\theta \ell_{\hat{S}}(\hat{\theta}_{\hat{S}}; D_i)\right]_j & \text{otherwise} \end{cases}$$

Alongside computational benefits, reducing the dimension of the data used for ACV from n to $|\hat{S}|$ also allows for more accurate approximation in higher dimensions (given we make a few assumptions on the data).

Similarly, the "sparse ACV" updates for IJ becomes,

$$\left[\tilde{\theta}_{\text{IJ}}^{-i}\right]_j = \begin{cases} 0 & \text{when } \hat{\theta}_j = 0 \\ \hat{\theta}_j + \left[\left(H_{\hat{S}}(\hat{\theta}_{\hat{S}}; D)\right)^{-1} \nabla_\theta \ell_{\hat{S}}(\hat{\theta}_{\hat{S}}; D_i)\right]_j & \text{otherwise} \end{cases}$$

Alongside computational benefits, reducing the dimension of the data used for ACV from n to $|\hat{S}|$ also allows for more accurate approximation in higher dimensions (given we make a few assumptions on the data). A major condition needed for this to work is the *incoherence condition* (also known as mutual incoherence). Where,

$$\max_{j \in S^C} \|(X_S^T X_S)^{-1} X_S^T X_j\|_1 \leq 1 - \gamma$$

for some $\gamma > 0$. This condition essentially uses the coefficients of a regression of X_S onto X_j to measure the alignment of the column X_j on X_S. In the ideal case, the columns in S^C are orthogonal to the columns in S ($\gamma = 1$) and we can recover the support (given more assumptions).

This method has also proved effective for non-simulated examples.

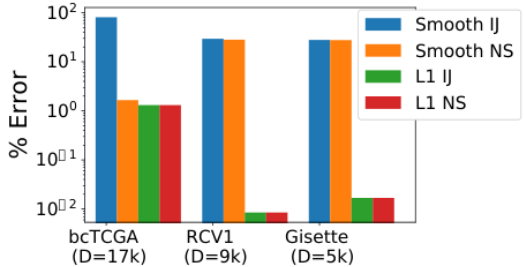

Figure 1: Error rates for smoothed and "sparse" (L1) ACV methods on real data (Stephenson and Broderick (2020)). Here the error is measured by $\text{Err} = \frac{\sum_{i=1}^{n} |\ell(\tilde{\theta}_{-i}; D) - \ell(\hat{\theta}_{-i}; D)|}{\sum_{i=1}^{n} |\ell(\hat{\theta}_{-i}; D)|}$

Section 3

Preliminary Work

Preliminary Work

I have already written working code recreating the experiments in the original IACV paper. This experiment is ACV applied to vanilla GD solving a logistic regression task.

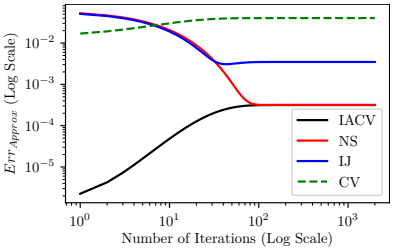

Figure 2: My implementation (run for less iterations).

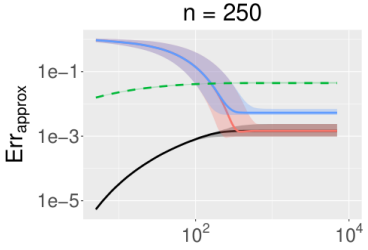

Figure 3: Experiment in the paper.

Here, $\text{Err}_{\text{Approx}} = \frac{1}{n}\sum_{i=1}^{n} \|\tilde{\theta}_{-i}^{(k)} - \hat{\theta}_{-i}^{(k)}\|_2^2$ and legends are shared.

I've implemented the experiments using explicit gradient calculation in Numpy and using automatic differentiation in JAX.

I've implemented the experiments using explicit gradient calculation in Numpy and using automatic differentiation in JAX. The JAX version also allows for experiments to be run on the GPU for faster processing - useful when data is large or in higher dimensions. I've also written a version in Pytorch (slightly slower than JAX due to inherent overhead) which can also work on the GPU.

I've implemented the experiments using explicit gradient calculation in Numpy and using automatic differentiation in JAX. The JAX version also allows for experiments to be run on the GPU for faster processing - useful when data is large or in higher dimensions. I've also written a version in Pytorch (slightly slower than JAX due to inherent overhead) which can also work on the GPU.

The JAX version specifically makes use of explicit vectorisation in approximate CV and shows a large possible empirical speed gain over explicit LOOCV due to the semantics of JAX vmap. Also, the functions for both the Jacobian and Hessian can be Just In Time (JIT) compiled for a boost in speed.

I've implemented the experiments using explicit gradient calculation in Numpy and using automatic differentiation in JAX. The JAX version also allows for experiments to be run on the GPU for faster processing - useful when data is large or in higher dimensions. I've also written a version in Pytorch (slightly slower than JAX due to inherent overhead) which can also work on the GPU.

The JAX version specifically makes use of explicit vectorisation in approximate CV and shows a large possible empirical speed gain over explicit LOOCV due to the semantics of JAX vmap. Also, the functions for both the Jacobian and Hessian can be Just In Time (JIT) compiled for a boost in speed.

There are also experimental subsets of Pytorch and JAX implementing sparse tensors to save space in high dimensional tasks, which will serve helpful to look into.

A small example of JAX features applied to IACV,

```
def F(theta, X, y, lbd):
    return jnp.sum(l(X, y, theta)) + lbd * pi(theta)

nabla_F = jit(grad(F))
hess_F = jit(jacfwd(jacrev(F)))

grad_per_sample = jit(vmap(nabla_F, in_axes=(None, 0, 0, None)))
hess_per_sample = jit(vmap(hess_F, in_axes=(None, 0, 0, None)))
vmap_matmul = jit(vmap(jnp.matmul, in_axes=(0, 0)))
```

here we vectorise per-sample gradient calculation and speed up Jacobian and Hessian evaluation through JIT.

In exact LOOCV, we have to resort to

```
for i in range(n):
    theta_true[i] = theta_true[i] - alpha *
        nabla_F(
        np.delete(X, (i), axis=0),
        np.delete(y, (i), axis=0),
        lbd=lbd_v)
```

as we cannot vectorise over variable shaped matrices.

In exact LOOCV, we have to resort to

```
for i in range(n):
    theta_true[i] = theta_true[i] - alpha *
        nabla_F(
        np.delete(X, (i), axis=0),
        np.delete(y, (i), axis=0),
        lbd=lbd_v)
```

as we cannot vectorise over variable shaped matrices.

Also - I may have found a bug in JAX. Numpy's delete is orders of magnitude faster than JAX's equivalent, this needs fixing.

Basic experiment for $n = 200$ and $p = 400$ as a test of IACV using a logistic LASSO.

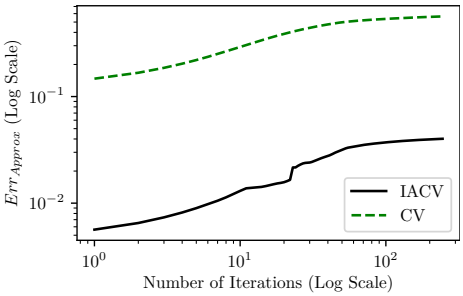

Figure 4: Preliminary high-dimensional test

Section 4

Future Plans

Future Plans

For the short term,

Future Plans

For the short term,

- Write a general ACV implementation (focused mainly on IACV) to wrap existing optimisation schemes in Python. The main goal is to wrap JAXopt and/or pytorch.optim optimiser objects to provide a low-cost simultaneous ACV step alongside the main optimisation step.

Future Plans

For the short term,

- Write a general ACV implementation (focused mainly on IACV) to wrap existing optimisation schemes in Python. The main goal is to wrap JAXopt and/or pytorch.optim optimiser objects to provide a low-cost simultaneous ACV step alongside the main optimisation step.
- Understand the theoretical bounds on IACV accuracy and convergence.

Future Plans

For the short term,

- Write a general ACV implementation (focused mainly on IACV) to wrap existing optimisation schemes in Python. The main goal is to wrap JAXopt and/or pytorch.optim optimiser objects to provide a low-cost simultaneous ACV step alongside the main optimisation step.
- Understand the theoretical bounds on IACV accuracy and convergence.
- Implement a basic version of the sparse ACV algorithm described in Stephenson and Broderick (2020). There is already code for this out there.

Future Plans

For the short term,

- Write a general ACV implementation (focused mainly on IACV) to wrap existing optimisation schemes in Python. The main goal is to wrap JAXopt and/or pytorch.optim optimiser objects to provide a low-cost simultaneous ACV step alongside the main optimisation step.
- Understand the theoretical bounds on IACV accuracy and convergence.
- Implement a basic version of the sparse ACV algorithm described in Stephenson and Broderick (2020). There is already code for this out there.

For the long term, there are two paths to go down:

- Adapt the implementation and theory of sparse ACV for IACV.

Future Plans

For the short term,

- Write a general ACV implementation (focused mainly on IACV) to wrap existing optimisation schemes in Python. The main goal is to wrap JAXopt and/or pytorch.optim optimiser objects to provide a low-cost simultaneous ACV step alongside the main optimisation step.
- Understand the theoretical bounds on IACV accuracy and convergence.
- Implement a basic version of the sparse ACV algorithm described in Stephenson and Broderick (2020). There is already code for this out there.

For the long term, there are two paths to go down:

- Adapt the implementation and theory of sparse ACV for IACV.
- Possibly look at different learning algorithms to apply IACV to. The main target here is soft-margin SVM.