EECS189 Course Notes

Anmol Parande

Fall 2020 - Professors Anant Sahai, Jitendra Malik, Jennifer Listgarten

Disclaimer: These notes reflect EECS189 when I took the course (Fall 2020). They may not accurately reflect current course content, so use at your own risk. If you find any typos, errors, etc, please raise an issue on the GitHub repository.

Contents

1 Introduction to Supervised Machine Learning

The basic problem of machine learning is to understand patterns from data and use these patterns to make predictions. In general, Machine Learning problems have three different components:

1. **Data:** A collection of (x_i, y_i) points.
2. **Model:** A pattern that we are looking for.
3. **Learning Algorithm:** A process to learn our pattern. This is split further into:
   (a) **Optimization Problem:** We find our pattern by minimizing (or maximizing) something.
   (b) **Optimization Algorithm:** How will we perform the minimization.

Definition 1 *Supervised learning is a subclass of machine learning where we try to map inputs $\vec{x_i}$ to labels y_i.*

In supervised learning, we are successful if given a new vector $\vec{x}$ that we have not seen at training time, we can predict the output $\hat{y}$ correctly. Supervised learning is broken down into two different subclasses based on the type of the label $\hat{y}$.

- **Regression Problems** The labels y or $\vec{y}$ are continuous.
- **Classification Problems** The labels y are categorial variables.

Machine Learning Models rarely use the data points x_i directly. We instead do our learning on features $\phi(x)$.

Definition 2 *Features are constituent elements of the learned patterns. This means we believe our output y is a linear combination of features.*

$$\vec{y} = \sum w_i \phi_i(\vec{x})$$

If $\vec{w} \in \mathbb{R}^m$ and $\vec{x} \in \mathbb{R}^r$, then we call it **lifting** when $m > r$ (more features than input dimensions) and **distillation** when $m < r$ (fewer features than input dimension). ML practitioners choose the features they use for models using domain-specific knowledge. One way we can select our features is using the idea of **Universal Function Approximation**.i

Definition 3 *A universal function family is a family of functions which allows (with enough features) arbitrarily good approximations of target patterns.*

Smooth functions can be expressed as a polynomial (Taylor Expansion) or as a sum of complex exponentials (Fourier Series). This is why polynomial features and fourier features are examples of universal function families. One downside to such families is that in the vector case, the number of features required scales very quickly, and this can impact our algorithm's ability to learn.

1.1 Overview of Classification

During classification, we need to encode the categories as numbers. One way which we do this is through **One-Hot Encoding**.

Definition 4 *A one-hot encoded vector $\vec{y}$ looks like*

$$\vec{y} = \begin{bmatrix} 1 \textit{ if class 1 else } 0 \\ 1 \textit{ if class 2 else } 0 \\ 1 \textit{ if class 3 else } 0 \\ 1 \textit{ if class 4 else } 0 \\ \vdots \end{bmatrix}$$

Because of this, we can really see classification as a form of regression since we have datapoints $(\vec{x_i}, \vec{y_i})$ where the $\vec{y_i}$ are just one-hot encodings. One approach to do classification then becomes to learn a weight vector for each of the l classes $\vec{w}_l$. Once we have these features, our prediction $\hat{y}$ is

$$\hat{y} = \underset{l}{\operatorname{argmin}} \, \vec{x}^T \vec{w}_l.$$

In the binary case, we know that

$$\vec{x}^T \vec{w}_1 > \vec{x}^T \vec{w}_2 \implies \vec{x}^T (\vec{w}_1 - \vec{w}_2 > 0),$$

if the first class is the prediction, which means that we only need to learn a single weight vector $\vec{w} = \vec{w}_1 - \vec{w}_2$.

1.2 Probabalistic Perspective

One way to understand machine learning is to look at it through the lens of probability. In particular, we have some test distribution (X_{test}, y_{test}) that we are trying to learn. Our training data (X_{train}, y_{train}) is taken from a training distribution that may or not be the same as the test distribution. Under this perspective,

$$y = \phi(x) w^* + N$$

where N is the observation noise. The denoised observations $\tilde{y} = \phi(x)w^*$. When using the probabalistic perspective, it is important to keep track of what is random and what isn't. For example, if our observations x_i are sampled at regular spacings, then X_{train} is not random. If there is observation noise, then y_{train} is random. Regardless, our prediction $\hat{w}(X_{train}, y_{train})$ is a function of random things (the training process, y_{train}, etc).

With the probabalistic perspective, there are two possible goals of learning:

1. **Parameter Recovery**: We want $\hat{w}$ to be close to w^* (i.e $\mathbb{E}\left[\|\hat{w} - w^*\|^2\right]$)
2. **Prediction Accuracy**: We want $\hat{y}_{test}$ close to $\tilde{y}_{test}$ (i.e $\mathbb{E}\left[\|\hat{y}_{test} - \tilde{y}_{test}\|^2\right]$).

It turns out that these two objectives are related. Starting with the prediction accuracy (assuming the noise is 0 mean and independent of the other variables),

$$\begin{aligned}
\mathbb{E}\left[(y_{test} - \hat{y}_{test})^2\right] &= \mathbb{E}\left[(\phi(X_{test})w^* + N - \phi(X_{test})\hat{w})^2\right] \\
&= \mathbb{E}\left[N^2\right] + \mathbb{E}\left[(w^* - \hat{w})^T\phi(X_{test})^T\phi(X_{test})(w^* - \hat{w})\right] \\
&= Var\left(N\right) + \mathbb{E}\left[(w^* - \hat{w})^T\Sigma_\phi(w^* - \hat{w})\right] \qquad \text{where } \Sigma_\phi = \mathbb{E}\left[\phi(X_test)^T phi(X_test)\right]
\end{aligned}$$

Σ_ϕ is the covariance of the test features. What this means is if $\Sigma_\phi = I$, then $\mathbb{E}\left[\|w^* - \hat{w}\|^2\right] = \mathbb{E}\left[\|\hat{y} - \tilde{y}\|\right]$ In English, this means the features are orthonormal with respect to the test distribution. The implication of this is that the test distribution of our data actually matters during learning!

2 Ridge Regression

2.1 Ordinary Least Squares

One of the most basic algorithms in machine learning is ordinary least squares. We have a weight vector $\vec{w}$ that we want to learn, a data matrix X, and a target vector $\vec{y}$ that we want to predict. The least squares solution is simply

$$\hat{w} = (X^TX)^{-1}X^T\vec{y}$$

However, this can be problematic. If we look at the SVD of $(X^TX)^{-1}$, we see that it is $(V\tilde{\Sigma}^2V^T)^{-1}$ where $\tilde{\Sigma}$ is the diagonal matrix of the singular values σ_i. If there are small σ_i, which represent insignificant directions in our data matrix, then a small change to the imput could have large impacts on the output (since $\frac{1}{\sigma_i}$ would become large).

2.1.1 The Optimization Perspective

The optimization problem whose solution is least squares is:

$$\hat{w} = \underset{w}{\operatorname{argmin}} \, ||\vec{y} - X\vec{w}||^2$$

2.1.2 The Probability Perspective

We can also view Least Squares from a probabalistic perspective as the Maximum Likelihood estimate. The MLE are the parameters which would maximize our probability of seeing the data we have. We assume that

$$Y_i = h(x_i) + z_i \qquad z_i \sim \mathcal{N}(0, \sigma^2)$$

In other words, we have fixed (non-random) x_i as our input data and outputs $Y_i \sim \mathcal{N}(f(X_i)$. The MLE estimate is

$$\underset{\vec{w}}{\text{argmax}} \prod_{i=1}^{n} f_y(Y = y_i | X = x_i; \hat{w})$$

Lets assume that $h(x_i) = x_i * w_i$. Since $Y_i \sim \mathcal{N}(\phi(x_i), \sigma^2)$,

$$f_y(Y = y | X = x; w_i) = \frac{1}{2\pi\sigma^2} e^{-\frac{(y - x_i w_i)^2}{2\sigma^2}}$$

$$\begin{aligned}
\underset{\vec{w}}{\text{argmax}} \prod_{i=1}^{n} f_y(Y = y_i | X = x_i; \hat{w}) &= \underset{\vec{w}}{\text{argmax}} \sum_{i=1}^{n} \ln(f_y(Y = y_i | X = x_i; w_i)) \\
&= \underset{\vec{w}}{\text{argmax}} -\frac{1}{2} \sum_{i=1}^{n} (y_i - x_i w_i)^2 \\
&= \underset{\vec{w}}{\text{argmin}} \sum_{i=1}^{n} \|\vec{y} - \vec{w}^T \vec{x}
\end{aligned}$$

This is the least squares optimization problem, so MLE with Gaussian noise is the same as least squares!

2.2 The Hack Perspective

Since the OLS solution has the possibility of being unstable because of small singular values, one thing we could do is instead say

$$\hat{w} = (X^T X + \lambda I)^{-1} X^T \vec{y}.$$

Looking at the SVD of

$$(X^T X + \lambda I)^{-1} = V \begin{bmatrix} (\tilde{\Sigma}_r^2 + \lambda I)^{-1} & 0 \\ 0 & \frac{1}{\lambda} I \end{bmatrix} V^T$$

The diagonal entries of the matrix $(\tilde{\Sigma}_r^2 + \lambda I)^{-1}$ are $\frac{1}{\sigma_j^2 + \lambda} = \frac{1}{\sigma_j^2} \left(\frac{1}{1 + \frac{1}{\sigma_j}^2} \right)$ Notice that this acts like a high pass filter. What this means is that small σ_j won't have as much as an effect because they are dominated by λ. The interpretation of this is that ridge regression is a soft way of expressing trust in the directions v_j (eigenvector of $X^T X$). To choose λ, we want to choose it such that $\lambda < \sigma_i^2$ for the directions which are trustworthy and $\lambda > \sigma_i^2$ for the directions that we do not trust.

2.3 The Optimization Perspective

To formalize our conception of ridge regression, we need to see if it solves some kind of optimization problem. It turns out that Ridge Regression is the solution to

$$\hat{w} = \underset{\vec{w}}{\operatorname{argmin}} \|\vec{y} - X\vec{w}\|^2 + \lambda\|\vec{w}\|^2$$

Proof 1 *We take the derivative and set it equal to zero.*

$$\begin{aligned} \|\vec{y} - X\vec{w}\|^2 + \lambda\|\vec{w}\|^2 = \vec{w}^T X^T X\vec{w} - 2\vec{w}^T X^T\vec{y} + \vec{y}^T\vec{y} + \lambda\vec{w}^T\vec{w} \\ \therefore 2X^T X\vec{w} - 2X^T\vec{y} + 2\lambda\vec{w} = 0 \\ \therefore \vec{w} = (X^T X + \lambda I)^{-1} X^T\vec{y} \end{aligned}$$

2.4 The Fake Data Perspective

One different way we can interpret Ridge regression is that we add fake data points and take the OLS solution. Consider the following data matrix and target vector given the original data X and target $\vec{y}$.

$$\tilde{X} = \begin{bmatrix} X \\ \sqrt{\lambda I} \end{bmatrix} \qquad \tilde{y} = \begin{bmatrix} \vec{y} \\ 0 \end{bmatrix}$$

If we apply OLS, we see that

$$(\tilde{X}^T\tilde{X})^{-1}\tilde{X}^T\tilde{y} = (X^T X + \lambda I)^{-1} X^T\vec{y}$$

This is identical to the ridge solution.

2.5 The Fake Feature Perspective

We can also achieve the same effect by adding fake features to each data point instead of adding fake data points. In other words, we have the following data matrix and weight vector.

$$\hat{X} = \begin{bmatrix} X & \alpha I \end{bmatrix} \qquad \vec{\eta} = \begin{bmatrix} \vec{w} \\ \epsilon \end{bmatrix}$$

This gives us a minimum norm problem

$$\min_{\vec{\eta}} \|\vec{\eta}\|^2 \text{ s.t } \hat{X}\vec{\eta} = \vec{y}$$

We can expand this out to

$$\min_{\vec{w},\epsilon} \|\vec{w}\|^2 + \|\epsilon\|^2 \text{ s.t} X\vec{w} + \alpha\epsilon = \vec{y}$$

We can see that $\epsilon = \frac{1}{\alpha}(\vec{y} - X\vec{w})$ which is a scaled version of the least squares residual. If we rescale the objective function, we get

$$\operatorname*{argmin}_{\vec{w}} \alpha^2\|\vec{w}\|^2 + \|\vec{y} - X\vec{w}\|^2$$

If we let $\alpha = \sqrt{\lambda}$, then we recover the ridge solution.

2.6 The Probability Perspective

Just like OLS, we can interpret Ridge Regression probabalistically. Recall the optimization problem for ridge regression is

$$\hat{w} = \operatorname*{argmin}_{\vec{w}} \|\vec{y} - X\vec{w}\|^2 + \lambda\|\vec{w}\|^2$$

Since we are penalizing the norm of the weights, this is equivalent to saying that we believe we have small weights. In probability, beliefs are expressed via priors. We assume that $w_i \sim \mathcal{N}(0, \beta)$ and use Maximum A-Posteriori Estimation (MAP) to compute the optimal $\vec{w}$.

$$\begin{aligned}
\operatorname*{argmax}_{\vec{w}} f(\vec{w}|data) &= \operatorname*{argmax}_{\vec{w}} \frac{f(y = y_{train}|X = x_{train}, w)f(\vec{w}|X_{train})}{f(Y = y_{train})} \\
&= \operatorname*{argmax}_{\vec{w}} f(Y = y_{train}|X = X_{train}, \vec{w})f(\vec{w}) \\
&= \operatorname*{argmax}_{\vec{w}} \sum_i \ln(f(y_i|w_i, x_i)) + \sum_j \ln(f(w_j)) \\
&= \operatorname*{argmin}_{\vec{w}} \|\vec{y} - X\vec{w}\|^2 + \frac{1}{\beta^2}\|\vec{w}\|^2
\end{aligned}$$

This is the optimization problem for ridge regression where $\lambda = \frac{1}{\beta^2}$. This has a very clean interpretation as well. If β^2 is large, λ is small. In other words, if we believe the paramters have wide spread, then we don't want to regularize as much. If β^2 is small, then λ is large, meaning if we believe our paramters are close to 0, we can safely regularize more.

3 Hyper-Paramters and Validation

3.1 Sources of Error

In Machine Learning, there are 5 qualitative sources of error.

3.1.1 True Parameter Errors

Definition 5 *Survival/Bias is how well the algorithm will do if there is no noise in the data. It is impacted heavily by the learning algorithm.*

Definition 6 *Variance is a measure of the impact of noise on the model's ability to recover the true parameter.*

To illustrate survival and variance, consider a scalar model $y = w^*x + N$ where $n \sim \mathcal{N}(0, \sigma^2)$. Given the training data (x_i, y_i), the Ridge Regression estimate is

$$\begin{aligned}\hat{w} = (\vec{x}^T\vec{x} + \lambda)^{-1}\vec{x}^T\vec{y} &= \frac{\vec{x}^T(w^*\vec{x} + \vec{N})}{\|\vec{x}\|^2 + \lambda} \\ &= \frac{w^*\|x\|^2}{\|\vec{x}\|^2 + \lambda} + \frac{\vec{x}^T\vec{N}}{\|\vec{x}\|^2 + \lambda}\end{aligned}$$

If there was no noise (i.e N=0), then our true weight would be shrunk by a factor of $\frac{\|\vec{x}\|^2}{\|\vec{x}\|^2+\lambda}$ (a.k.a only a certain percent of our true weight survives the training process). With noise, the learned weight becomes a random variable.

$$\hat{w} \sim \mathcal{N}\left(\frac{\|\vec{x}\|^2}{\|\vec{x}\|^2 + \lambda}, \frac{\sigma^2}{\|\vec{x}\|^2\left(1 + \frac{\lambda}{\|\vec{x}\|^2}\right)}\right)$$

3.1.2 External Contamination

Definition 7 *External contamination is when our model learns features that don't belong in the model, and those features also affect how the model uses the true feature.*

To see how external contamination works, consider an example where we we have the test distribution $Y = w_1^*X[1] + N$. In other words, we have vector features $\vec{X}$, but the output is a noisy, scaled version of it. For simplicity, we also assume that our features are orthonormal to each other. If we use Ridge Regression to estimate the true parameter, we get

$$\hat{w} = (X^TX)^-1X^T\vec{w} = X^T(w_1^*X[1] + N) = \begin{bmatrix} w_1^* \\ 0 \\ \vdots \\ 0 \end{bmatrix} + X^TN$$

Clearly, the weights $\hat{w}_j, j \neq 1$ are non-zero, and the weight $\hat{w}_1$ is contaminated with some noise of the other features.

The amount of external contamination present depends on the learning algorithm used. Suppose that we don't have orthonormal training features and our true model is $\vec{y} = w_1^* \vec{x}_1$. In other words, our output is a noiseless scaled version of the first feature. If we use OLS,

$$\hat{w} = (X^T X)^{-1} X^T \vec{w} = (X^T X)^{-1} X^T w_1^* \vec{x}_1 = w_1^* (X^T X)^{-1} X^T \vec{x}_1 = w_1^* \vec{e}_1$$

we get no contamination. However, if we use Ridge Regression,

$$\hat{w} = w_1^* (X^T X + \lambda I)^{-1} X^T \vec{x}_1 = \begin{bmatrix} w_1^* \cdot SV \\ * \\ \vdots \\ * \end{bmatrix} \neq w_1^* \vec{e}_1$$

where SV is the survival factor and the $*$ are junk. Recall there is no noise in this example, meaning our contamination truly came from the training, not from the noise.

3.1.3 Approximation Error

Definition 8 *Approximation error is mismatch between the model and the true pattern because the model is not complex enough to represent the true pattern accurately*

Suppose the true pattern is

$$f(x) = \sum_{j=1}^{m} w^*[j]\phi_j(x) + \sum_{j=m+1}^{k} w^*[j]\phi_j(x)$$

If we only include m features, then the learned pattern will be

$$f(x) = \sum_{j=1}^{m} \hat{w}^*[j]\phi_j(x) + \sum_{j=m+1}^{k} \hat{w}^*[j]\phi_j(x)$$

The best we can possibly do is

$$f(x) = \sum_{j=1}^{m} w^*[j]\phi_j(x)$$

This means our prediction error

$$\mathbb{E}\left[(y - \hat{y})^2\right] = \sum_{j=m+1}^{k} \hat{w}^*[j]\phi_j(x)$$

This goes down as we increase m (i.e add more features)

3.1.4 Irreducible Error

Definition 9 *Irreducible Error is the prediction error that is entirely out of our control due to noise/corruption in the data*

Suppose we have the test distribution $Y = w^*X + N$ where $N \sim \mathcal{N}(0, \sigma^2)$ and X is orthonormal to the test distribution (i.e $\mathbb{E}[X^2] = 1$). Given training points (x_i, y_i), then our prediction error becomes

$$\begin{aligned}\mathbb{E}\left[(Y - \hat{w}X)^2\right] | &= \mathbb{E}\left[((w^* - \hat{w})X + N)^2\right] \\ &= \sigma^2 + \mathbb{E}\left[(w^* - \hat{w})^2 X^2\right] \\ &= \sigma^2 + \mathbb{E}\left[(w^* - \hat{w})^2\right]\end{aligned}$$

Clearly, no matter how well we recover the true parameter, we are still going to have at least σ^2 error because of the noise in the test distribution, and this error is irreducible and independent of the learning algorithm/model we use.

1. **True Parameter Errors**
2. **External Contamination:** We learn features part of the pattern that don't actually belong to the true pattern.
3. **Approximation Error:** Mismatch between model and the true pattern. Suppose the true pattern is
4. **Irreducible Error:** Truly unpredictable error that we can do nothing about.

3.2 Hyper-Paramters

Definition 10 *A hyper-paramter is a tunable paramter of our model that is not being optimized by our learning algorithm.*

Definition 11 *A model hyperparameter is one which determines the structure of the model*

An example of a model-hyperparamter is the number of polynomial features we use for ridge regression. It isn't something which we optimize directly, but it affects our model (the $\vec{w}$).

Definition 12 *An optimization hyperparamter is a paramter which impacts the structure of the optimization problem.*

An example of an optimization hyperparamter is the reguarlization constant in Ridge Regression.

3.3 Validation

One big question is how do we optimally set the hyper-paramters of our model. We don't have the true function, so we need to use data to evaluate how good our model is at predicting new data.

Definition 13 *A test set is data not part of the training data that is used to evaluate the model quality.*

If we are evaluating the model in a mean squared error sense, then our quality metric is

$$\frac{1}{n_{test}} \sum_{i=1}^{n_{test}} (\hat{y}(X_{test,i}) - y_{test,i})^2$$

We hope that the noise in y_{test} doesn't matter (i.e it does not conspire to defeat our model). However, we can't still can't choose our paramters using this test set because otherwise we are optimizing our model for this particular test set, but we still have no idea if it will work for other test sets.

Definition 14 *A validation set is a subset of the training data kept aside to optimize hyperparamters*

One way of doing validation is to split our data (80-20 train-test and 70-30 train-validation is normal) and iterate through different hyperparameter combinations.

Listing 1: Validation Pseudocode

```
Outer Loop: Optimize hyperparamters based on validation error
  Inner Loop: Optimize paramters based on training data, holding
     hyperparamters constant
```

In this way, our model depends on the validation and train data. This is why it is crucial to hold the test data separate. Notice that this method of validation is not very data efficient because it cuts the number of points we use to train the paramters. One technique that helps this is K-fold cross validation.

Listing 2: K-Fold Cross Validation Pseudocode

```
Split data into K folds
Train K times:
  Choose one fold for validation and the rest is training data
Compute the average validation error
```

K-fold cross validation fits into the inner loop of listing 1.

4 Kernels

4.1 The Training Data Perspective

Suppose we have training data X and outputs $\vec{y}$ as the data. We featurize the training data into a matrix Φ where each column of Φ is a different feature $\Phi_i(\vec{x_i})$. Then the estimated parameter $\hat{w}$ in ridge regression is a linear function in $\vec{y}$. At prediction time, we get a test point $\vec{x}$ and our prediction becomes

$$\hat{y} = \sum_{j=1}^{n} \left(\Phi(\vec{x})(\Phi^T\Phi + \lambda I)^{-1}\Phi^T[j]\right) y_j = \sum_{j=1}^{n} \hat{h}_j(\vec{x}) y_j$$

where $\hat{h}_j$ is a scalar function of $\vec{x}$ and has no dependence on $\vec{y}$. It captures the influence of the jth training point on the predictions. The problem is that $\hat{h}_j$ is not very interpretable. Ideally, we would like something like Lagrange Interpolation for polynomials $\sum_{i=1}^{n} y_j L_j(x)$ where

$$L_j(x) = \frac{\prod_{j=1}^{n}(x - x_j)}{\prod_{i \neq j}(x_i - x_j)}$$

Notice that the numerator expresses the similarity between the new point x and the points x_j which have already been seen whereas the denominator expresses the relationship similarity between pairwise points that we have already seen. To try and tease out this relationship for Ridge Regression, we can shift perspective to the fake data perspective where the solution becomes

$$\hat{w} = \Phi^T(\Phi\Phi^T + \lambda I)^{-1}\vec{y}$$

Notice that the matrix $K = \Phi\Phi^T$ is a matrix of pairwise inner products (Gram matrix), and thus it captures the relationship between the features of each data point. Looking at our prediction,

$$\hat{y} = \Phi(\vec{x})^T\hat{w} = \sum_{j=1}^{n} y_i \hat{h}_j(\vec{x})$$

$$\hat{h}_j = \sum_{i=1}^{n} \left(< \Phi_j(\vec{x}_i), \Phi_j(\vec{x}) > (K + \lambda I)^{-1}[j]\right)[i]$$

This gives us a view like Lagrange Interpolation since $< \Phi_j(\vec{x}_i), \Phi_j(\vec{x}) >$ relates old points to the new ones and $(K + \lambda I)^{-1}[j]$ reflects the relationship amongst the training data $\vec{x}_j$. Alternatively,

$$\hat{y} = \sum_{i=1}^{n} < \Phi(\vec{x}_i), \Phi(\vec{x}) > g[i]$$

$$g[i] = (K + \lambda I)^{-1}\vec{y}$$

where $g[i]$ captures the influence of the ith training point.

4.2 Similarity Kernels

The key realization to arrive at kernels is that we never actually need the features $\Phi(\vec{x_i})$ of our training data in order to do prediction. Instead, we can just compute inner products using a function.

Definition 15 *A Kernel function $K(\vec{x}, \vec{z})$ is an inner product with a positive semi-definite Gram matrix.*

$$K(\vec{x}, \vec{z}) =< \Phi(\vec{x}), \Phi(\vec{z}) > \qquad K = \begin{bmatrix} K(\vec{x_1}, \vec{x_1}) & K(\vec{x_1}, \vec{x_2}) & \cdots & K(\vec{x_1}, \vec{x_n}) \\ \vdots & \ddots & & \vdots \\ K(\vec{x_n}, \vec{x_1}) & K(\vec{x_n}, \vec{x_2}) & \cdots & K(\vec{x_n}, \vec{x_n}) \end{bmatrix}$$

A valid kernel also implies that there is a set of Φ_i features. As it turns out, the number of features required to represent a kernel need not be finite! Below are two examples of popular similarity kernels.

Polynomial Kernel (Degree p): $K_p(\vec{x}, \vec{z}) = (1 + \vec{x}^T \vec{z})^p$

Radial Basis Function (RBF): $K(\vec{x}, \vec{z}) = e^{\frac{-\|\vec{x}-\vec{z}\|^2}{2\sigma^2}}$

4.3 Influence Kernels

We can also engineer directly at the level of $\hat{f}(\vec{x}) = \sum_{i=1}^{n} y_i \hat{h}_i(\vec{x})$ where $\hat{h}_i(\vec{x}) = h(\vec{x} - \vec{x}_i)$. This function $\hat{h}$ is known as an influence kernel, and depending on the kernel function, it can be thought of as a nearest neighbors approach.

5 Regularizing Effect of Features

From the feature augmentation view of ridge regression,

$$\underset{\vec{w}}{\operatorname{argmin}} \|\vec{w}\|^2 + \|\vec{\eta}\|^2 \text{ s.t } \begin{bmatrix} X & \sqrt{\lambda} I \end{bmatrix} \begin{bmatrix} \vec{w} \\ \vec{\eta} \end{bmatrix} = \vec{y}$$

The $\sqrt{\lambda} I$ features are the Ridge features, and they act differently based on train/test time.

- At test time, they predict 0 (i.e they are non-contaminating)
- At train time, they are very localized and only predict $\sqrt{\lambda}$ for the given training point.

Intuitively, this means the ridge weights η absorb some of the energy of $\vec{y}$ in a harmless way. If instead, the ridge features predict something small at test time, this would still be relatively harmless. If instead our ridge features were not $\sqrt{\lambda}I$ but some other unitary transform, this interpretation would also still largely stay the same. More formally, what this means is that **extra features have regularizing abilities**.

6 Latent Spaces

6.1 PCA

In problems where we have a data-matrix X and an output vector $\vec{y}$, the structure of X is very important. If we interpret the SVD of X

$$X = \sum_i \sigma_i \vec{u}_i \vec{v}_i^T$$

the row vectors $\vec{v}_i^T$ are independent directions in the feature space, and in order for features not to regularize our model, we need to the correct σ_i to be large enough to overcome their effect. PCA gives us a way to identify and focus on these important directions. If we order our singular values in decreasing order, then we can create synthetic features

$$XV = U\Sigma = [\sigma_1 \vec{u}_1 \ \sigma_2 \vec{u}_2 \ \cdots \ \sigma_m \vec{u}_m]$$

where the $\sigma_i \vec{u_i}$ (the left singular vectors) are the synthetic features. By the Eckart-Young-Mirsk Theorem, we know that the best rank-k approximation to X (in the $\|\cdot\|_F$ sense) is $\sum_{i=1}^k \sigma_i \vec{u}_i \vec{v}_i^T$. What this means is that our feature matrix X is really just a linear transformation of a latent space whose basis is the first k $\vec{u}_k$. The other's are a basis for some irrelevant subspace that we do not want to learn. To understand this better, consider an example with Gaussian features (i.e $\|\vec{x}_i\| 0 \Sigma_x$) where $\Sigma_x = V\Lambda V^T$ and $\vec{u}_i = \Lambda^{-\frac{1}{2}} V^T \vec{x}_i$.

$$\begin{aligned} \mathbb{E}[\vec{u}] &= 0 \\ \mathbb{E}\left[U\vec{U}^T\right] &= \Lambda^{-\frac{1}{2}} V^T \Sigma_x V \Lambda^{-\frac{1}{2}} = I \\ \therefore U &\sim \mathcal{N}(0, I_m) \end{aligned}$$

If we now imagine that our X are there because of feature noise and an irrelevant subspace, then

$$\Lambda^{\frac{1}{2}} = \begin{bmatrix} \sqrt{\sigma_1^2 - \sigma_{low}^2} & & 0 \\ & \ddots & \\ 0 & & \sqrt{\sigma_l^2 - \sigma_{low}^2} \end{bmatrix}$$

Observe two things:

1. Feature noise can be regularizing since effectively add $\sigma_{low}^2 I$ to some true $\tilde{X}$. This effectively asks learning to be robust to variation
2. We can get the same regularization as ridge regression by doing PCA followed by OLS
   (A) Do PCA and get the top k synthetic features
   (B) Use OLS to predict y based on synthetic features
   (C) Use cross-validation to pick K

This will work assuming that the directions of most variation are also the ones which are most relevant to making good predictions.

A Multi-variate Gaussians

A multi-variate Gaussian random variable is specified by its mean $\vec{\mu}_x$ and covariance matrix Σ_x. The covariance matrix specifies the covariance between each pair of elements in the random variable.

$$\vec{X} \sim \mathcal{N}(\vec{\mu}_X, \Sigma_X) \qquad \mathbb{E}\left[\vec{X}\right] = \mu_X$$

$$Cov\left(\vec{X}\right) = \mathbb{E}\left[(\vec{X} - \vec{\mu}_X)(\vec{X} - \vec{\mu}_X)^T\right]$$

Each element of the covariance matrix is $\Sigma_X[i,j] = Cov\left(\vec{X}[i]\vec{X}[j]\right)$.

Theorem 1 *If we have a multivariate gaussian random variable which is jointly distributed (i.e, Σ is not diagonal), then $\exists \vec{W}$ with identically and indepdently distributed elements such that $\vec{W} \sim \mathcal{N}(0, I)$ and $\exists A$ such that $\vec{X} = \vec{\mu}_X + A\vec{w}$.*

Proof 2 *We can quickly see that*

$$Cov\left(\vec{X}\right) = \mathbb{E}\left[A\vec{W}\vec{W}^T A^T\right] = AA^T = \Sigma_X$$

Since Σ_X is a real, symmetric matrix, we can orthogonally diagonalize it as $\Sigma_X = V\Lambda V^T$. To see how we compute A, first consider the vector random variable $G_i = \vec{v}_i\vec{X}$.

$$\begin{aligned} Var\left(G_i\right) &= \mathbb{E}\left[(\vec{v}_i\vec{X} - \vec{v}_i^T\vec{\mu}_X)^2\right] \\ &= \vec{v_i}^T AA^T\vec{v_i} = \vec{v_i}^T\Sigma_X\vec{v_i} \\ &= \vec{v_i}V\Lambda V^T\vec{v_i} = \lambda_i \end{aligned}$$

. This means that $\lambda \geq 0$ since variances are always non-negative, and consequently that Σ_X is positive, semi-definite. This means we can define A to be the symmetric square root of Σ_X. In other words,

$$A = V\sqrt{\Lambda} \text{ or } A = V\sqrt{\Lambda}V^T$$

Now, provided that none of the X_i in $\vec{X}$ are linearly dependent (so no zero eigenvalues, we can whiten $\vec{W}$ into $\vec{W} \sim \mathcal{N}(0, I)$.

$$\vec{W} = A^{-1}(\vec{X} - \vec{\mu}_X)$$

The PDF of the $\vec{W}_i$ is

$$f_w(w_i) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}w_i^2}$$

This makes the PDF of the entire vector

$$f_{\vec{w}} = \prod_{i=1}^{n} f_w(w_i) = \frac{1}{(2\pi)^{\frac{n}{2}}} e^{-\frac{1}{2}\vec{w}^T\vec{w}}$$

Now if we want to find the PDF of the original random variable $\vec{X} = A\vec{W}$,

$$f_X(\vec{X}) = \frac{1}{|A|} f_{\vec{w}}(A^{-1}(\vec{x} - \vec{\mu}_X)) = \frac{1}{\sqrt{(2\pi)^n |\Sigma_X|}} e^{-\frac{1}{2}(\vec{x}-\vec{\mu}_x)^T\Sigma_x^{-1}(\vec{x}-\vec{\mu}_X)}$$

The determinant appears as a rescaling factor to make sure the function integrates to 1.