# CPSC 340 Assignment 5 (due Friday November 25 at 11:55pm)

Name(s) and Student ID(s):

## 1 Kernel Trick and SGD

In this question you will revisit questions from previous assignments, this time implementing the same (or similar) models using the "kernel trick" and SGD.

### 1.1 "Other" Normal Equations

The script *example_nonLinear* loads a dataset from a previous assignment, and fits an L2-regularized least squares model with a bias term (the regularization leads to a small improvement in the test error, even for this 2-variable problem). Modify the *leastSquaresBiasL2* function so that it uses the "other" normal equations we discussed in class,

$$v = Z^T \underbrace{(ZZ^T + \lambda I)^{-1} y}_{u}.$$

In particular, the training should result in an $n \times 1$ vetor $u$ of parameters, and predictions are made based on the training $Z$ and the vector $u$. Hand in your code for the modified function.

Hint: you should get the same predictions as the original funciton. To help debugging, you may want to first re-write the calculation of $v$ using the above formula, to verify that this gives you the same vector $v$.

Answer: Squared train Error with least squares: 3551.961
Squared test Error with least squares: 3393.095

```
function leastSquaresBiasL2(X,y,lambda)
        # Add bias column
        n = size(X,1)
        Z = [ones(n,1) X]

        # Find regression weights minimizing squared error
        v = Z'*inv(Z*Z' + lambda*I)*y

        # Make linear prediction function
        function predict(Xhat)
                Zhat = [ones(size(Xhat,1),1) Xhat]
                return (Zhat * Z') * inv(Z*Z' + lambda*I) * y
        end

        # Return model
        return LinearModel(predict,v)
end
```

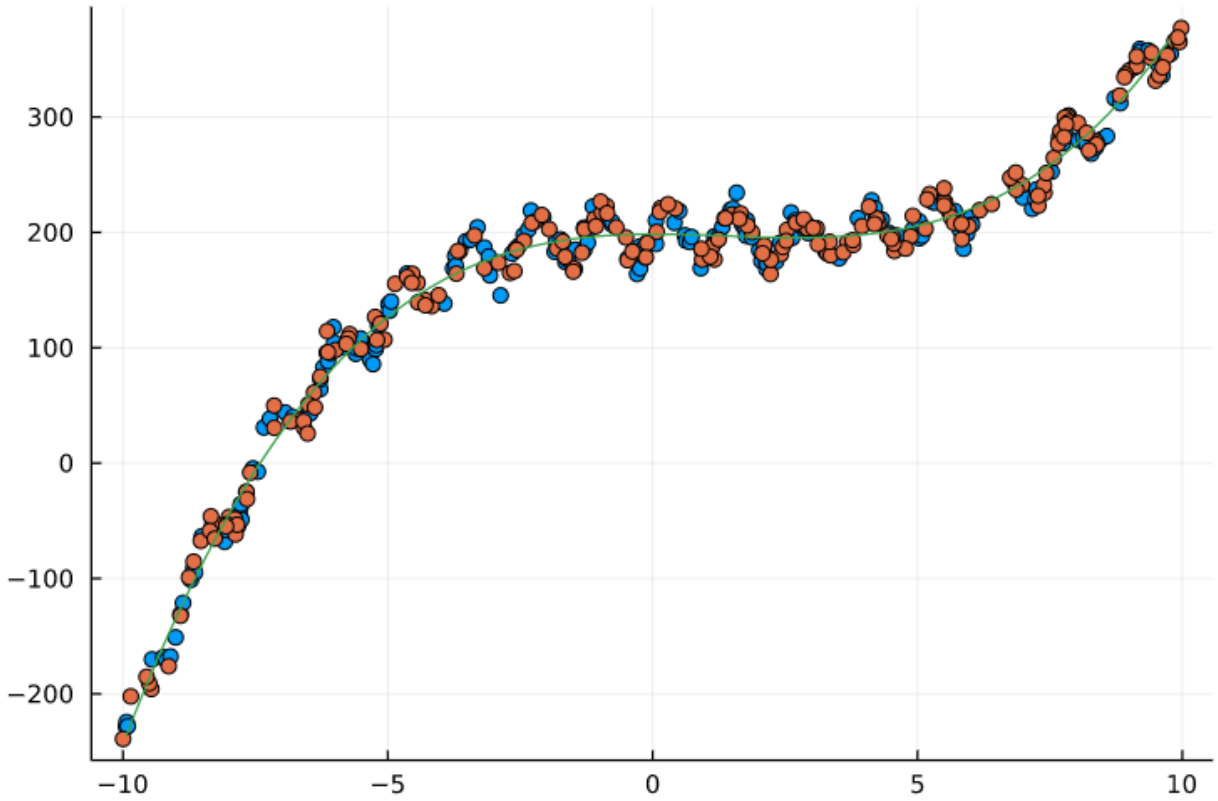
## 1.2 Polynomial Kernel

Write a new function, *leastSquaresKernelBasis*, taking a degree $p$ and a regularization parameter $\lambda$. It should fit a degree-$p$ polynomial to the data, using the kernel trick to avoid ever forming the matrix $Z$. Hand in your code and the plot obtained with $p = 3$ and $\lambda = 10^{-6}$.

Hint: you may find it helpful to write a function *polyKernel* that takes two matrices as inputs (either $X$ and $X$ or $\tilde{X}$ and $X$) and a degree $p$, and computes the polynomial kernel between all pairs of rows in the matrices.

Answer: Squared train Error with least squares: 252.021
Squared test Error with least squares: 242.790

```
function leastSquaresKernelBasis(X, y, p, lambda)
        n = size(X, 1)
        K = zeros(n, n)

        for i in 1:n
                for j in 1:n
                        K[i, j] = polyKernel(X[i, :], X[j, :], p)
                end
        end

        u = (K + lambda * I)\(y)

        function predict(Xhat)
                t = size(Xhat, 1)
                Khat = zeros(t, n)

                for i in 1:t
                        for j in 1:n
                                Khat[i, j] = polyKernel(Xhat[i, :], X[j, :], p)
                        end
                end

                return Khat * u
        end

        return LinearModel(predict, u)
end

function polyKernel(xi, xj, p)
        return (1 + xi' * xj)^p
end
```

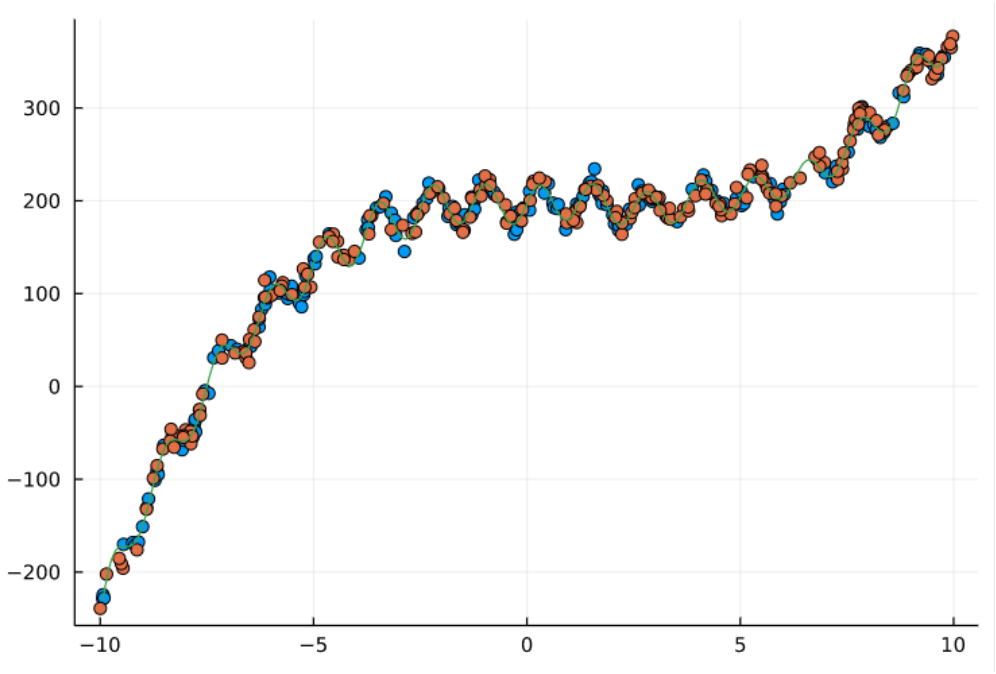

300
200
100
0
−100
−200
−10
−5
0
5
10

### 1.3 Gaussian-RBF Kernel

Repeat the previous question and report the same quantities, but using the Gaussian RBF kernel. You can use $\sigma = 1$ and $\lambda = 10^{-6}$ for the plot.

Answer: Squared train Error with least squares: 39.163
Squared test Error with least squares: 70.580

```
function leastSquaresKernelRBF(X,y,sigma,lambda)
        n = size(X, 1)

        K = zeros(n, n)

        for i in 1:n
                for j in 1:n
                        K[i, j] = RBFKernel(X[i, :], X[j, :], sigma)
                end
        end

        u = (K + lambda * I)\(y)

        function predict(Xhat)
                t = size(Xhat, 1)
                Khat = zeros(t, n)

                for i in 1:t
                        for j in 1:n
                                Khat[i, j] = RBFKernel(Xhat[i, :], X[j, :], sigma)
                        end
                end

                return Khat * u
        end

        return LinearModel(predict, u)
end

function RBFKernel(xi,xj,sigma)
        return exp(-norm(xi-xj)^2/(2sigma^2))
end
```


300
200
100
0
−100
−200
−10
−5
0
5
10

### 1.4 Stochastic Gradient Descent

Instead of using normal equations to fit a linear least squares model, the script *example_GD* fits the regularized linear model using gradient descent. It uses 500 iterations and a step size that guarantees that the regularized squared error decreases on each step. Modify this demo to implement stochastic gradient descent update, which on iteration $t$ uses

$$v^{t+1} = v^t - \alpha_t((v^T z_i - y_i)z_i + (\lambda/n)v),$$

where on each iteration $i$ is a random training example.

1. Hand in your code implementing the SGD update.
2. Consider step sizes of the form $\alpha_t = \gamma/t$, $\alpha_t = \gamma/\sqrt{t}$, and $\alpha_t = \gamma$ (for some constant $\gamma$). If we consider value of $\gamma$ that are powers of 10, and run SGD for 500 iterations with $\lambda = 1$, what value tends to give the best performance for each of these types of step sizes?
3. On this problem, which of the three types of step size do you think is best to use? And which is the worst to use?

Answer: 1.

```
for t in 1:500
        i = rand(1:n)
        global v -= alpha*((v'*Z[i, :]-y[i, :])[1]*Z[i,:] + (lambda/n)*v)
        @show (1/2)norm(Z*v-y)^2 + lambda*norm(v)^2
end
```

Answer: 2. Not sure why my gamma are returning NAN, which are infinities.

Answer: 3. I think the best step size to use would be $\alpha_t = \gamma/t$, and the worst step size to use is $\alpha_t = \gamma$.

## 2 MAP Estimation

In class, we considered MAP estimation in a regression model where we assumed that:

- The likelihood $p(y_i \mid x_i, w)$ for each example $i$ is a normal distribution with a mean of $w^T x_i$ and a variance of 1.
- The prior $p(w_j)$ for each variable $j$ is a normal distribution with a mean of zero and a variance of $\lambda^{-1}$.

Under these assumptions we showed that computing the MAP estimate with $n$ training examples leads to the standard L2-regularized least squares objective function:

$$f(w) = \frac{1}{2}\|Xw - y\|^2 + \frac{\lambda}{2}\|w\|^2.$$

For each of the alternate assumptions below, write down the objective function that results (from minimizing the negative log-posterior, and simplifying as much as possible):

1. We use a Laplace likelihood with a mean of $w^T x_i$ and a scale of 1, and we use a zero-mean Laplace prior for each variable with a scale parameter of $\lambda^{-1}$,

$$p(y_i \mid x_i, w) = \frac{1}{2}\exp(-|w^T x_i - y_i|), \quad p(w_j) = \frac{\lambda}{2}\exp(-\lambda|w_j|).$$

Answer: $p(w) = \Pi_{j=1}^{d} p(w_j)$
$= (\frac{\lambda}{2}^d) \cdot \exp(-\lambda \sum_{j=1}^{d} |w_j|$

$-log(p(w)) = -log(\frac{\lambda}{2}^d) + \lambda\|w\|_1$

$-log(p(y_i|x_i, w)) = \|Xw - y\|_1 - nlog(\frac{1}{2})$

$f(w) = -log(p(y_i|x_i, w) - log(p(w)$

$f(w) = \|Xw - y\|_1 + \lambda\|w\|_1 - dlog(\frac{\lambda}{2}) - nlog(\frac{1}{2})$

2. We use a normal likelihood with a mean of $w^T x_i$ but where each example $i$ has its own positive variance $\sigma_i^2$, and a normal prior with a variance of $\lambda^{-1}$ and a mean that is some "guess" $w^0$ of the optimal parameter vector,

$$p(y_i \mid x_i, w) = \frac{1}{\sqrt{2\sigma_i^2\pi}}\exp\left(-\frac{(w^T x_i - y_i)^2}{2\sigma_i^2}\right), \quad p(w_j) \propto \exp\left(-\frac{\lambda(w_j - w_j^0)^2}{2}\right).$$

The standard notation for this case is to use $\Sigma$ as a diagonal matrix with the $\sigma_i^2$ values along the diagonal.

Answer: $p(w) = \Pi_{j=1}^{d} p(w_j)$
$= \exp(-\frac{\lambda}{2}\sum_{j=1}^{d}(w_j - w_j^0)^2$

$-log(p(w)) = \frac{\lambda}{2}\|w - w^0\|^2$

$-log(p(y|x, w)) = (Xw - y)^T(\Sigma^{-1})(Xw - y) + nlog(\sqrt{2\pi}) + \sum_{i=1}^{n} log(|\sigma_i|)$
NOTE: $\Sigma^{-1}$ is representative of the inverse of $\Sigma$

$f(w) = -log(p(y|x, w) - log(p(w)$

$f(w) = (Xw - y)^T(\Sigma^{-1})(Xw - y) + \frac{\lambda}{2}\|w - w^0\|^2 + nlog(\sqrt{2\pi}) + \sum_{i=1}^{n} log(|\sigma_i|))$

3. We use a Poisson likelihood with a mean of $\exp(w^T x_i)$,[1] and we use a uniform prior for some constant $\kappa$,

$$p(y_i \mid x_i, w) = \frac{\exp(y_i w^T x_i)\exp(-\exp(w^T x_i))}{y_i!}, \quad p(w_j) \propto \kappa$$

For this sub-question you don't need to put likelihood in matrix notation.

Answer: $p(w) = \Pi_{j=1}^{d} p(w_j)$
$= d\kappa$

$-log(p(w)) = -log(d\kappa)$

$-log(p(y_i|x_i, w)) = -\sum_{i=1}^{n}\left(log(\exp(y_i w^T x_i) + log(\exp(-\exp(w^T x_i)) - log(y_i!)\right)$

$= -y^T Xw + \sum_{i=1}^{n}(\exp(w^T x_i)) + \sum_{i=1}^{n} log(y_i!)$

$f(w) = -y^T Xw + \sum_{i=1}^{n}(\exp(w^T x_i)) + \sum_{i=1}^{n} log(y_i!) - log(d\kappa)$

4. We use a Laplace likelihood with a mean of $w^T x_i$ where each example $i$ has its own positive scale paramater $v_i^{-1}$, and a student $t$ prior (which is very robust to irrelevant features) with $\nu$ degrees of freedom,

$$p(y_i \mid x_i, w) = \frac{1}{2}\exp\left(-v_i|w^T x_i - y_i|\right), \quad p(w_j) = \frac{\Gamma\left(\frac{\nu+1}{2}\right)}{\sqrt{\nu\pi}\Gamma\left(\frac{\nu}{2}\right)}\left(1 + \frac{w_j^2}{\nu}\right)^{-\frac{\nu+1}{2}}$$

where you use can $V$ as a diagonal matrix with the $v_i$ along the diagonal and $\Gamma$ is the "gamma" function (which is always non-negative). You do not need to put the log-prior in matrix notation.

Answer: $-log(p(w)) = -log(\Pi_{j=1}^{d} p(w_j))$

$-log(p(y_i|x_i, w)) = -nlog(\frac{1}{2}) + \|V(Xw - y)\|_1$

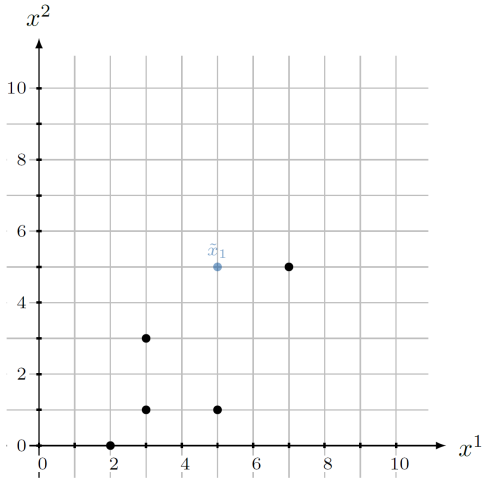
$f(w) = -nlog(\frac{1}{2}) + \|V(Xw - y)\|_1 - log(\Pi_{j=1}^{d} p(w_j))$

[1] This is one way to use regression to model *counts*, like "number of Facebook likes".

# 3 Principal Component Analysis (PCA)

## 3.1 PCA by Hand

Consider the following dataset, containing $n = 5$ training examples and $t = 1$ test example with $d = 2$ features each:




| | $x^1$ | $x^2$ |
|---|---|---|
| | 2 | 0 |
| | 3 | 1 |
| $X$ | 7 | 5 |
| | 3 | 3 |
| | 5 | 1 |
| $\widetilde{X}$ | 5 | 5 |

1. Principal component analysis requires a centered feature matrix. Find the vector of means $\mu = \begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}$ and compute the centered feature matrix $X_c$.

2. Writing $G_c = \begin{bmatrix} g_{11} & g_{12} \\ g_{21} & g_{22} \end{bmatrix} = X_c^\top X_c$, the first principal component $w_1 = \begin{bmatrix} w_{11} \\ w_{12} \end{bmatrix}$ is given by the **normalized** solution to the linear system $(G_c - \mathbf{I}_d \lambda_1) \cdot w_1 = 0$, where $\lambda_1$ is the largest solution to the quadratic equation $\chi(\lambda) = (\lambda - g_{11})(\lambda - g_{22}) - g_{12} g_{21} = 0$.[2] The second principal component $w_2 = \begin{bmatrix} w_{21} \\ w_{22} \end{bmatrix}$ is given by the **normalized** solution to the linear equation $w_1^\top w_2 = 0$. Compute the two principal components $w_1$ and $w_2$ (show your work).

3. The two principal components $w_1$ and $w_2$ form an orthonormal basis $W = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix}$ of $\mathbb{R}^2$. Represent the test point $\tilde{x}_1$ in this basis by performing an orthogonal projection $\tilde{z}_1 = \begin{bmatrix} \tilde{z}_{11} \\ \tilde{z}_{12} \end{bmatrix} = W(\tilde{x}_1 - \mu)$ onto $w_1$ and $w_2$ (show your work).

[2] $\chi$ is called the *characteristic polynomial* of $G_c$ and its roots are eigenvalues $\lambda$ whose corresponding eigenvectors $w$ solve the equation $G_c w = \lambda w$.

4. We can now represent the PCA coordinate system in the orginal coordinate system. In the plot above, draw the two coordinate axes $5w_1$ and $5w_2$, using $\mu$ as origin. Then visualize the PCA coordinates $\tilde{z}_1$ computed in 3.1.3 by drawing two lines between the test point and its orthogonal projections onto $w_1$ (given by $\begin{bmatrix} \tilde{z}_{11} \\ 0 \end{bmatrix}$ in the PCA coordinate system) and $w_2$ (given by $\begin{bmatrix} 0 \\ \tilde{z}_{12} \end{bmatrix}$ in the PCA coordinate system). Finally, use the visualization to read off the coordinates of $\begin{bmatrix} \tilde{z}_{11} \\ 0 \end{bmatrix}$ and $\begin{bmatrix} 0 \\ \tilde{z}_{12} \end{bmatrix}$ in the **original** coordinate system, which can be interpreted as reconstructions of $\tilde{x}_1$. Report both reconstructions and their L2 reconstruction errors (show your work).

   Note: If you want, you can draw the PCA coordinate system and the orthogonal projections directly in Latex; see the commented lines below this one in a5.tex for an example. Hand-drawn results will also be accepted.

### 3.2 Data Visualization

The script *example_PCA* will load a dataset containing 50 examples, each representing an animal. The 85 features are traits of these animals. The script also shows a heatmap of the data matrix $X$. Although it possible to try to interpret this heatmap, it is not as interpretable as a scatterplot.

The function *PCA* applies the classic PCA method (orthogonal bases via SVD) for a given $k$. Using this function, modify the demo to output a scatterplot of the latent features $z_i$ from the PCA model with $k = 2$. You can use the *scatter* function from *Plots.jl* to output a scatterplot. Use the *annotate!* function to label the points in the scatter plot with the animal names (I set *annotationfontsize=8* when calling this function).

1. Hand in your modified demo and the scatterplot which has the annotated animal names.

2. For each of the first three principal components answer the following: which animals are furthest apart according to this principal component, and which trait of the animals has the largest influence (absolute value) on the first principal component? (Make sure not to forget the "+1" when looking for the name of the animal/trait in the *dataTable*).

### 3.3 Data Compression

It is important to know how much of the information in our dataset is captured by the low-dimensional PCA representation. In class we discussed the "analysis" view that PCA maximizes the variance that is explained by the PCs, and the connection between the Frobenius norm and the variance of a centered data matrix $X$. Use this connection to answer the following:

1. How much of the variance is explained by our two-dimensional representation from the previous question?
2. How many PCs are required to explain 75% of the variance in the data?

Note: you can compute the Frobenius norm of a matrix using the function *norm*. Also, note that the "variance explained" formula from class assumes that $X$ is already centered.

# 4 Very-Short Answer Questions

1. What is the difference between multi-label and multi-class classification?
2. We discussed "global" vs. "local" features for e-mail classification. What is an advantage of using global features, and what is advantage of using local features?
3. Describe the kernel trick in L2-regularized least squares in one sentence.
4. What is the key advantage of stochastic gradient methods over gradient descent methods?
5. Which of the following step-size sequences lead to convergence of stochastic gradient to a stationary point?
   (a) $\alpha^t = 1/t^2$.
   (b) $\alpha^t = 1/t$.
   (c) $\alpha^t = \gamma/t$ (for $\gamma > 0$).
   (d) $\alpha^t = 1/(t+\gamma)$ (for $\gamma > 0$).
   (e) $\alpha^t = 1/\sqrt{t}$.
   (f) $\alpha^t = 1$.
6. Given discrete data $D$ and parameters $w$, is the expression $p(D|w)$ a probability mass function, a likelihood function, or both? Briefly justify your answer.
7. Why is it often more convenient to minimize the negative log-likelihood than to maximize the likelihood?
8. How does the impact of the prior in MAP estimation change as we add more data?
9. What is the difference between a generative model and a discriminative model?
10. With PCA, is it possible for the loss to increase if $k$ is increased? Briefly justify your answer.
11. Why doesn't it make sense to do PCA with $k > d$?
12. In terms of the matrices associated with PCA ($X$, $W$, $Z$, $\hat{X}$), where would an "eigenface" be stored?