CPSC 340 Assignment 5 (due Monday March 18 at 11:55pm)

Instructions

Rubric: {mechanics:5}

IMPORTANT!!! Before proceeding, please carefully read the general homework instructions at `https://www.cs.ubc.ca/~fwood/CS340/homework/`. The above 5 points are for following the submission instructions. You can ignore the words "mechanics", "reasoning", etc.

We use blue to highlight the deliverables that you must answer/do/submit with the assignment.

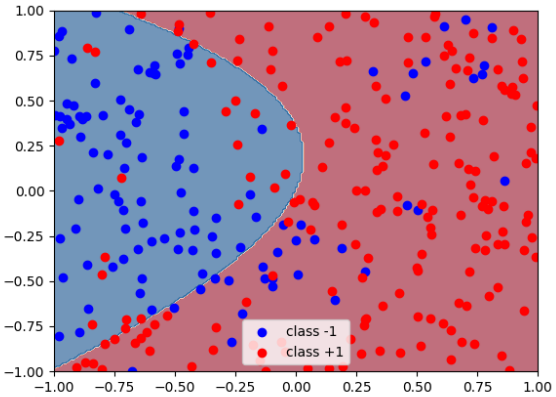
1 Kernel Logistic Regresion

If you run `python main.py -q 1` it will load a synthetic 2D data set, split it into train/validation sets, and then perform regular logistic regression and kernel logistic regression (both without an intercept term, for simplicity). You'll observe that the error values and plots generated look the same since the kernel being used is the linear kernel (i.e., the kernel corresponding to no change of basis).

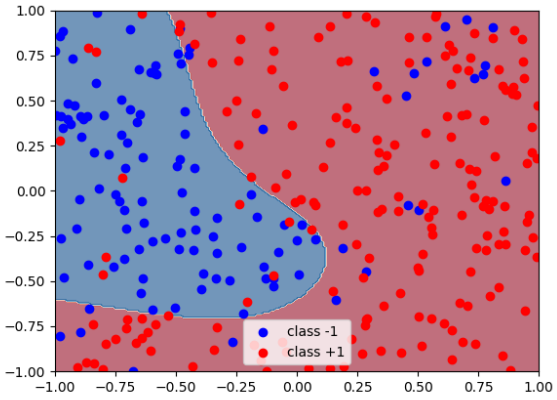
1.1 Implementing kernels

Rubric: {code:5}

Implement the polynomial kernel and the RBF kernel for logistic regression. Report your training/validation errors and submit the plots generated for each case. You should use the hyperparameters $p = 2$ and $\sigma = 0.5$ respectively, and $\lambda = 0.01$ for the regularization strength.
Polynomial Kernel:
Training error 0.183
Validation error 0.170



RBF Kernel:
Training error 0.127
Validation error 0.090



In LogReg.py:

```
def kernel_RBF(X1, X2, sigma=1):
    return np.exp((-1) * utils.euclidean_dist_squared(X1, X2) / (2 * np.power(sigma, 2)))

def kernel_poly(X1, X2, p=2):
    return (1 + X1@X2.T) ** p
```

In main.py:

```
elif question == "1.1":
        dataset = load_dataset('nonLinearData.pkl')
        X = dataset['X']
        y = dataset['y']

        Xtrain, Xtest, ytrain, ytest = train_test_split(X,y,random_state=0)

        # YOUR CODE HERE
        pl_kernel = kernelLogRegL2(kernel_fun=logReg.kernel_poly, lammy=0.01, p=2)
        pl_kernel.fit(Xtrain, ytrain)

        print("Training error %.3f" % np.mean(pl_kernel.predict(Xtrain) != ytrain))
        print("Validation error %.3f" % np.mean(pl_kernel.predict(Xtest) != ytest))

        utils.plotClassifier(pl_kernel, Xtrain, ytrain)
        utils.savefig("logRegPolyKernel.png")

        rbf_kernel = kernelLogRegL2(kernel_fun=logReg.kernel_RBF, lammy=0.01, sigma=0.5)
        rbf_kernel.fit(Xtrain, ytrain)

        print("Training error %.3f" % np.mean(rbf_kernel.predict(Xtrain) != ytrain))
        print("Validation error %.3f" % np.mean(rbf_kernel.predict(Xtest) != ytest))
```

```
utils.plotClassifier(rbf_kernel, Xtrain, ytrain)
utils.savefig("logRegRBFKernel.png")
```

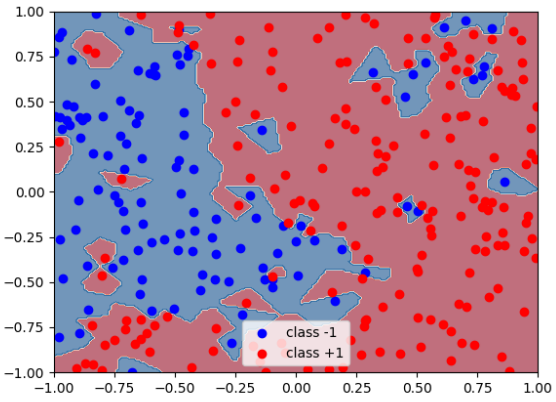
1.2 Hyperparameter search

Rubric: {code:3}

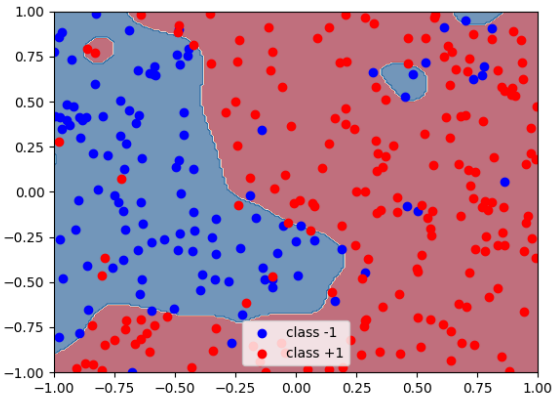
For the RBF kernel logistic regression, consider the hyperparameters values $\sigma = 10^m$ for $m = -2, -1, \ldots, 2$ and $\lambda = 10^m$ for $m = -4, -3, \ldots, 0$. In `main.py`, sweep over the possible combinations of these hyperparameter values. Report the hyperparameter values that yield the best training error and the hyperparameter values that yield the best validation error. Include the plot for each.

Note: on the job you might choose to use a tool like scikit-learn's `GridSearchCV` to implement the grid search, but here we are asking you to implement it yourself by looping over the hyperparameter values.
Best training error hyperparameters : $\sigma = 0.01$ $\lambda = 0.0001$
Best validation error hyperparameters : $\sigma = 0.1$ $\lambda = 1$

Best Training error:



Best Validation error:



```
elif question == "1.2":
        dataset = load_dataset('nonLinearData.pkl')
        X = dataset['X']
        y = dataset['y']

        Xtrain, Xtest, ytrain, ytest = train_test_split(X,y,random_state=0)

        # YOUR CODE HERE
        bestTrain = np.inf
        bestValidate = np.inf
        bestTrainSigma = 0
        bestTrainLammy = 0
        bestValidateSigma = 0
        bestValidateLammy = 0
        for i in range(-2, 3):
            for j in range(-4, 1):
                rbf_kernel = kernelLogRegL2(kernel_fun=logReg.kernel_RBF, lammy=10 ** j, sigma=10 ** i)
                rbf_kernel.fit(Xtrain, ytrain)
                if np.mean(rbf_kernel.predict(Xtrain) != ytrain) < bestTrain:
                    bestTrainSigma = 10 ** i
                    bestTrainLammy = 10 ** j
                    bestTrain = np.mean(rbf_kernel.predict(Xtrain) != ytrain)
                if np.mean(rbf_kernel.predict(Xtest) != ytest) < bestValidate:
                    bestValidateSigma = 10 ** i
                    bestValidateLammy = 10 ** j
                    bestValidate = np.mean(rbf_kernel.predict(Xtest) != ytest)
        print("Best training error hyperparameters :", bestTrainSigma, bestTrainLammy)
        print("Best validation error hyperparameters :", bestValidateSigma, bestValidateLammy)
        bestTrainModel = kernelLogRegL2(kernel_fun=logReg.kernel_RBF, lammy=bestTrainLammy, sigma=bestT
        bestTrainModel.fit(Xtrain, ytrain)
        utils.plotClassifier(bestTrainModel, Xtrain, ytrain)
        utils.savefig("bestTrainModel.png")
        bestValidateModel = kernelLogRegL2(kernel_fun=logReg.kernel_RBF, lammy=bestValidateLammy, sigma=
        bestValidateModel.fit(Xtrain, ytrain)
        utils.plotClassifier(bestValidateModel, Xtrain, ytrain)
        utils.savefig("bestValidateModel.png")
```

1.3 Reflection

Rubric: {reasoning:1}

Briefly discuss the best hyperparameters you found in the previous part, and their associated plots. Was the training error minimized by the values you expected, given the ways that σ and λ affect the fundamental tradeoff?

For Gaussian RBF, the narrower the bumps the more complicated the model. This means that lower σ value leads to lower training error and higher approximation error. Indeed, the model with the best training error chose a smaller σ than the model with the best validation error. In addition, the higher the regularization strength the higher the training error and the lower the validation error. Indeed, the model with the best training error chose the lowest λ possible while the model with the best validation error chose the highest λ possible.

2 MAP Estimation

Rubric: {reasoning:8}

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

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

Under these assumptions, we showed that this leads to the standard L2-regularized least squares objective function,

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

which is the negative log likelihood (NLL) under these assumptions (ignoring an irrelevant constant). For each of the alternate assumptions below, show how the loss function would change (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 Gaussian prior with a variance of σ^2.

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

$f(w) = -\sum_{i=1}^n log(p(y_i \mid x_i, w)) - \sum_{j=1}^d log(p(w_j))$
$= -\sum_{i=1}^n log(\frac{1}{2}\exp(-|w^T x_i - y_i|)) - \sum_{j=1}^d log(\frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{w_j^2}{2\sigma^2}\right))$
$= -\sum_{i=1}^n [log(\frac{1}{2}) + log(\exp(-|w^T x_i - y_i|))] - \sum_{j=1}^d [log(\frac{1}{\sqrt{2\pi}\sigma}) + log(\exp\left(-\frac{w_j^2}{2\sigma^2}\right))]$
$= -\sum_{i=1}^n [constant - |w^T x_i - y_i|] - \sum_{j=1}^d [constant - \frac{w_j^2}{2\sigma^2}]$
$= \|Xw - y\|_1 + \frac{1}{2\sigma^2}\|w\|^2$

2. We use a Gaussian likelihood where each datapoint has its own variance σ_i^2, and where we use a zero-mean Laplace prior with a vairance of λ^{-1}.

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

You can use Σ as a diagonal matrix that has the values σ_i^2 along the diagonal.
$f(w) = -\sum_{i=1}^n log(p(y_i \mid x_i, w)) - \sum_{j=1}^d log(p(w_j))$
$= -\sum_{i=1}^n log(\frac{1}{\sqrt{2\sigma_i^2\pi}}\exp\left(-\frac{(w^T x_i - y_i)^2}{2\sigma_i^2}\right)) - \sum_{j=1}^d log(\frac{\lambda}{2}\exp(-\lambda|w_j|))$
$= -\sum_{i=1}^n [log(\frac{1}{\sqrt{2\sigma_i^2\pi}}) + log(\exp\left(-\frac{(w^T x_i - y_i)^2}{2\sigma_i^2}\right))] - \sum_{j=1}^d [log(\frac{\lambda}{2}) + log(\exp(-\lambda|w_j|))]$
$= -\sum_{i=1}^n [constant - \frac{(w^T x_i - y_i)^2}{2\sigma_i^2}] - \sum_{j=1}^d [constant - \lambda|w_j|]$
$= \frac{1}{2}(Xw - y)^T\Sigma^{-1}(Xw - y) + \lambda\|w\|_1$

3. We use a (very robust) student t likelihood with a mean of $w^T x_i$ and ν degrees of freedom, and a zero-mean Gaussian prior with a variance of λ^{-1},

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

where Γ is the "gamma" function (which is always non-negative).

$f(w) = -\sum_{i=1}^{n} log(p(y_i \mid x_i, w)) - \sum_{j=1}^{d} log(p(w_j))$

$= -\sum_{i=1}^{n} log(\frac{\Gamma(\frac{\nu+1}{2})}{\sqrt{\nu\pi}\Gamma(\frac{\nu}{2})}\left(1 + \frac{(w^T x_i - y_i)^2}{\nu}\right)^{-\frac{\nu+1}{2}}) - \sum_{j=1}^{d} log(\frac{\sqrt{\lambda}}{\sqrt{2\pi}}\exp\left(-\lambda\frac{w_j^2}{2}\right))$

$= -\sum_{i=1}^{n}[log(\frac{\Gamma(\frac{\nu+1}{2})}{\sqrt{\nu\pi}\Gamma(\frac{\nu}{2})}) + log(\left(1 + \frac{(w^T x_i - y_i)^2}{\nu}\right)^{-\frac{\nu+1}{2}}))] - \sum_{j=1}^{d}[log(\frac{\sqrt{\lambda}}{\sqrt{2\pi}}) + log(\exp\left(-\lambda\frac{w_j^2}{2}\right)]$

$= -\sum_{i=1}^{n}[constant - (\frac{v+1}{2})log\left(1 + \frac{(w^T x_i - y_i)^2}{\nu}\right)] - \sum_{j=1}^{d}[constant - \lambda\frac{w_j^2}{2}]$

$= (\frac{v+1}{2})\sum_{i=1}^{n} log\left(1 + \frac{(w^T x_i - y_i)^2}{\nu}\right) + \frac{\lambda}{2}\|w\|^2$

4. We use a Poisson-distributed likelihood (for the case where y_i represents counts), and we use a uniform prior for some constant κ,

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

(This prior is "improper" since $w \in \mathbb{R}^d$ but it doesn't integrate to 1 over this domain, but nevertheless the posterior will be a proper distribution.)

$f(w) = -\sum_{i=1}^{n} log(p(y_i \mid x_i, w)) - \sum_{j=1}^{d} log(p(w_j))$

$= -\sum_{i=1}^{n} log(\frac{\exp(y_i w^T x_i)\exp(-\exp(w^T x_i))}{y_i!}) - \sum_{j=1}^{d} log(\kappa)$

$= -\sum_{i=1}^{n}[-log(y_i!) + log(\exp(y_i w^T x_i - \exp(w^T x_i)))] - \sum_{j=1}^{d} log(\kappa)$

$= -\sum_{i=1}^{n}[constant + y_i w^T x_i - \exp(w^T x_i)] - constant$

$= \sum_{i=1}^{n}(\exp(w^T x_i) - y_i w^T x_i)$

3 Principal Component Analysis

Rubric: {reasoning:3}

Consider the following dataset, containing 5 examples with 2 features each:

x_1	x_2
-4	3
0	1
-2	2
4	-1
2	0

Recall that with PCA we usually assume that the PCs are normalized ($\|w\| = 1$), we need to center the data before we apply PCA, and that the direction of the first PC is the one that minimizes the orthogonal distance to all data points.

1. What is the first principal component?
   The mean of the first variable is $(-4 + 0 - 2 + 4 + 2)/5 = 0$ and the mean of the second variable is $(3 + 1 + 2 - 1 + 0)/5 = 1$. Hence, after centering, the data becomes:

x_1	x_2
-4	2
0	0
-2	1
4	-2
2	-1

The direction of the line is $(2, -1)$ and after normalization the first component $w_1 = (\frac{2}{\sqrt{5}}, \frac{-1}{\sqrt{5}})$

2. What is the reconstruction loss (L2 norm squared) of the point (-3, 2.5)? (Show your work.)
To get a low-dimension representation, we subtract the means and multiply by w_c:
$z = \frac{2}{\sqrt{5}}(-3-0) + \frac{-1}{\sqrt{5}}(2.5-1) = \frac{-6}{\sqrt{5}} - \frac{1.5}{\sqrt{5}} = \frac{-7.5}{\sqrt{5}}$
To get back to the original space, we multiply this by w_c and add back the means:
$\hat{x} = \frac{-7.5}{\sqrt{5}}(\frac{2}{\sqrt{5}}, \frac{-1}{\sqrt{5}}) + (0, 1) = (-3, 2.5)$
Then, the reconstruction loss is:
$(-3-(-3))^2 + (2.5-2.5)^2 = 0$

3. What is the reconstruction loss (L2 norm squared) of the point (-3, 2)? (Show your work.)
To get a low-dimension representation, we subtract the means and multiply by w_c:
$z = \frac{2}{\sqrt{5}}(-3-0) + \frac{-1}{\sqrt{5}}(2-1) = \frac{-6}{\sqrt{5}} - \frac{1}{\sqrt{5}} = \frac{-7}{\sqrt{5}}$
To get back to the original space, we multiply this by w_c and add back the means:
$\hat{x} = \frac{-7}{\sqrt{5}}(\frac{2}{\sqrt{5}}, \frac{-1}{\sqrt{5}}) + (0, 1) = (-2.8, 2.4)$
Then, the reconstruction loss is:
$(-3-(-2.8))^2 + (2-2.4)^2 = 0.04 + 0.16 = 0.2$

Hint: it may help (a lot) to plot the data before you start this question.

4 PCA Generalizations

4.1 Robust PCA

Rubric: {code:10}

If you run `python main -q 4.1` the code will load a dataset X where each row contains the pixels from a single frame of a video of a highway. The demo applies PCA to this dataset and then uses this to reconstruct the original image. It then shows the following 3 images for each frame:

1. The original frame.
2. The reconstruction based on PCA.
3. A binary image showing locations where the reconstruction error is non-trivial.

Recently, latent-factor models have been proposed as a strategy for "background subtraction": trying to separate objects from their background. In this case, the background is the highway and the objects are the cars on the highway. In this demo, we see that PCA does an OK job of identifying the cars on the highway in that it does tend to identify the locations of cars. However, the results aren't great as it identifies quite a few irrelevant parts of the image as objects.

Robust PCA is a variation on PCA where we replace the L2-norm with the L1-norm,

$$f(Z, W) = \sum_{i=1}^{n}\sum_{j=1}^{d} |\langle w^j, z_i\rangle - x_{ij}|,$$

and it has recently been proposed as a more effective model for background subtraction. Complete the class *pca.RobustPCA*, that uses a smooth approximation to the absolute value to implement robust PCA. Briefly comment on the results.

Note: in its current state, *pca.RobustPCA* is just a copy of *pca.AlternativePCA*, which is why the two rows of images are identical.

Hint: most of the work has been done for you in the class *pca.AlternativePCA*. This work implements an alternating minimization approach to minimizing the (L2) PCA objective (without enforcing orthogonality). This gradient-based approach to PCA can be modified to use a smooth approximation of the L1-norm. Note that the log-sum-exp approximation to the absolute value may be hard to get working due to numerical issues, and a numerically-nicer approach is to use the "multi-quadric" approximation:

$$|\alpha| \approx \sqrt{\alpha^2 + \epsilon},$$

where ϵ controls the accuracy of the approximation (a typical value of ϵ is 0.0001).
The RobustPCA separates the cars with the highway much better than the AlternativePCA for most of the frames. That is, there are less traces of cars in $\hat{x}(L1)$ than $\hat{x}(L2)$ and the cars show up clearer in the L1 version than the L2 version.

```
class RobustPCA(AlternativePCA):
    def fit(self, X):
        n,d = X.shape
        k = self.k
        self.mu = np.mean(X,0)
        X = X - self.mu

        # Randomly initial Z, W
        z = np.random.randn(n*k)
        w = np.random.randn(k*d)
        for i in range(10): # do 10 "outer loop" iterations
            z, f = findMin(self._fun_obj_z, z, 10, w, X, k)
            w, f = findMin(self._fun_obj_w, w, 10, z, X, k)
            print('Iteration %d, loss = %.1f' % (i, f))

        self.W = w.reshape(k,d)

    def compress(self, X):
        n,d = X.shape
        k = self.k
        X = X - self.mu
        # We didn't enforce that W was orthogonal
        # so we need to optimize to find Z
        # (or do some matrix operations)
        z = np.zeros(n*k)
        z,f = findMin(self._fun_obj_z, z, 100, self.W.flatten(), X, k)
        Z = z.reshape(n,k)
        return Z

    def _fun_obj_z(self, z, w, X, k):
        n,d = X.shape
        Z = z.reshape(n,k)
        W = w.reshape(k,d)

        R = np.dot(Z,W) - X
        V = np.sqrt(R ** 2 + 0.0001)
        f = np.sum(V)
        DR = R / V
        g = np.dot(DR, W.transpose())
        return f, g.flatten()
```

```
def _fun_obj_w(self, w, z, X, k):
    n,d = X.shape
    Z = z.reshape(n,k)
    W = w.reshape(k,d)

    R = np.dot(Z, W) - X
    V = np.sqrt(R ** 2 + 0.0001)
    f = np.sum(V)
    DR = R / V
    g = np.dot(Z.transpose(), DR)
    return f, g.flatten()
```

4.2 Reflection

Rubric: {reasoning:3}

1. Briefly explain why using the L1 loss might be more suitable for this task than L2.
   Since the cars are outliers and the L1 loss is more more robust to outliers, it means that L1 "ignores" the cars better than that in L2. This is why the highways are much cleaner in the video frames.
2. How does the number of video frames and the size of each frame relate to n, d, and/or k?
   n is the number of video frames and the size of each frame is d (since d is the product of width and height of each frame) and k is unrelated to the number of frames or size of frame(It is the number of PCs).
3. What would the effect be of changing the threshold (see code) in terms of false positives (cars we identify that aren't really there) and false negatives (real cars that we fail to identify)?
   If we increase the threshold, we are going to have more false negatives and fewer false positives.

5 Very-Short Answer Questions

Rubric: {reasoning:11}

1. Assuming we want to use the original features (no change of basis) in a linear model, what is an advantage of the "other" normal equations over the original normal equations?
   The advantage is that it is faster if $n << d$ since the cost is now $O(n^2d + n^3)$ instead of $O(nd^2 + d^3)$
2. In class we argued that it's possible to make a kernel version of k-means clustering. What would an advantage of kernels be in this context?
   Kernel version of k-means clustering allows the clusters to be non-convex.
3. In the language of loss functions and regularization, what is the difference between MLE and MAP?
   Since MAP maximizes the posterior $p(w|D)$ whereas MLE maximizes the likelihood $p(D|w)$, this means that MAP has an extra prior. Thus, in the loss functions, MAP has regularization whereas MLE does not.
4. What is the difference between a generative model and a discriminative model?
   A generative model models $p(y_i, x_i)$, in another word, it models how the features X are generated. A discriminative model treats X as fixed and directly models $p(y_i|x_i)$.

5. With PCA, is it possible for the loss to increase if k is increased? Briefly justify your answer.
No it is not possible. In the worst case, the second component will just be the same line as the first component, so the loss is the same and not higher.

6. What does "label switching" mean in the context of PCA?
Since for 2 vectors w1 and w2 the $span(w1, w2) = span(w2, w1)$, we can switch the labels of w1 and w2.

7. Why doesn't it make sense to do PCA with $k > d$?
We use PCA for reducing dimensions so it doesn't make sense to reduce to a higher dimension.

8. In terms of the matrices associated with PCA (X, W, Z, $\hat{X}$), where would an "eigenface" be stored?
Since the "eigenface' are the principal components, they would be stored in W.

9. What is an advantage and a disadvantage of using stochatic gradient over SVD when doing PCA?
Advantage: With large datasets, SVD is too expensive whereas stochastic gradient is much cheaper(the cost per iteration is only $O(k)$).
Disadvantage: Stochastic gradient doesn't provide exact solutions whereas SVD does.

10. Which of the following step-size sequences lead to convergence of stochastic gradient to a stationary point?

    (a) $\alpha^t = 1/t^2$.
    No because the ratio $\frac{\sum_{t=1}^{\infty} \frac{1}{t^4}}{\sum_{t=1}^{\infty} \frac{1}{t^2}} = \frac{\frac{\pi^4}{90}}{\frac{\pi^2}{6}} \neq 0$

    (b) $\alpha^t = 1/t$.
    Yes because the ratio $\frac{\sum_{t=1}^{\infty} \frac{1}{t^2}}{\sum_{t=1}^{\infty} \frac{1}{t}} = 0$

    (c) $\alpha^t = 1/\sqrt{t}$.
    Yes because the ratio $\frac{\sum_{t=1}^{\infty} \frac{1}{t}}{\sum_{t=1}^{\infty} \frac{1}{\sqrt{t}}} = \frac{O(log(t))}{O(\sqrt{t})} = 0$

    (d) $\alpha^t = 1$.
    No, constant step size leads to erratic behavior inside the ball.

11. We discussed "global" vs. "local" features for e-mail classification. What is an advantge of using global features, and what is advantage of using local features?
The advantage of "global" features is that we can use it to predict what is important to a generic user.
The advantage of "local" features is that we can use it to make prediction personalized (important to a particular user).