CPSC 540 Assignment 2 (due February 12 at midnight)

The assignment instructions are the same as for the previous assignment, but for this assignment you can work in groups of 1-3. However, please only hand in one assignment for the group.

1. Name(s):
2. Student ID(s):

1 Convexity

1.1 From Definitions

Show that the following functions are convex, by only using one of the definitions of convexity. In other words, don't use the "operations that preserve convexity" or other convexity results stated in class,, but instead show that the function follows from one of the definitions of convexity we gave in class:[1]

1. Upper bound on second-order Taylor expansion: $f(w) = f(v) + \nabla f(v)^T(w-v) + \frac{L}{2}\|w-v\|^2$, for $L > 0$, which is often used in the analysis of gradient descent.
2. Maximum absolute value: $f(w) = \max_{j\in\{1,2,\ldots,d\}} |w_j|$.
3. The exponential loss, $f(w) = \sum_{i=1}^{n} \exp(-y^i w^\top x^i)$, used by AdaBoost.

Hint: Max and absolute value are not differentiable in general, so you cannot use the Hessian for functions involving those operations.

[1] That C^0 convex functions are below their chords, that C^1 convex functions are above their tangents, or that C^2 convex functions have a positive semidefinite Hessian.

1.2 Using Operations that Preserve Convexity

Show that the following functions are convex (you can use results from class and operations that preserve convexity if they help):

4. Support vector regression: $f(w) = \sum_{i=1}^{N} \max\{0, |w^\top x_i - y_i| - \epsilon\} + \frac{\lambda}{2}\|w\|_2^2$.
5. Mixed-norm regularization: $f(w) = \|w\|_{p,q} = \left(\sum_{g \in \mathcal{G}} \|w_g\|_q^p\right)^{\frac{1}{p}}$ (for $p \geq 1$ and $q \geq 1$, and $\mathcal{G}$ partitioning the variables into a set of "groups"). This is a generalization of L1-regularization that encourages sparsity in groups of variables (setting entire groups to 0). Hint: show that this is a norm.
6. Negative minimum of entropy over pairs of variables: $f(w) = -\min_{\{i,j \mid i \neq j\}}\{-w_i \log w_i - w_j \log w_j\}$ subject to $w_i > 0$ for all i.

1.3 Strong Convexity

Which of the functions 1-6 in the previous subsections are strongly convex?

2 Discrete and Gaussian Variables

2.1 MLE for General Discrete Distribution

Consider a density estimation task, where we have two variables ($d = 2$) that can each take one of k discrete values. For example, we could have

$$X = \begin{bmatrix} 1 & 3 \\ 4 & 2 \\ \text{k} & 3 \\ 1 & \text{k-1} \end{bmatrix}.$$

The likelihood for example x^i under a general discrete distribution would be

$$p(x_1^i, x_2^i \mid \Theta) = \theta_{x_1^i, x_2^i},$$

where θ_{c_1,c_2} gives the probability of x_1 being in state c_1 and x_2 being in state c_2, for all the k^2 combinations of the two variables. In order for this to define a valid probability, we need all elements θ_{c_1,c_2} to be non-negative and they must sum to one, $\sum_{c_1=1}^{k} \sum_{c_2=1}^{k} \theta_{c_1,c_2} = 1$.

1. Given n training examples, derive the MLE for the k^2 elements of Θ.
2. Note that this question was updated on February 4th. If we had separate parameters θ_{1,c_1} and θ_{2,c_2} for each variable, a standard for the prior would be a product of independent Dirichlet distributions,

$$p(\Theta) \propto \left[\prod_{c_1} \theta_{1,c_1}^{\alpha_{c_1}-1}\right]\left[\prod_{c_2} \theta_{c_2}^{\alpha_{c_2}-1}\right],$$

where in this case the two variables share k hyper-parameters $\alpha_1, \alpha_2, \ldots, \alpha_k$. Alternately, we could use a Dirichlet distribution over the k^2 variables,

$$p(\Theta) \propto \prod_{c_1=1}^{k} \prod_{c_2=1}^{k} \theta_{c_1,c_2}^{\alpha_{c_1,c_2}-1},$$

with k^2 hyper-parameters. An alternate approach that could be considered is to use

$$p(\Theta) \propto \prod_{c_1=1}^{k} \prod_{c_2=1}^{k} \theta_{c_1,c_2}^{\alpha_{c_1}+\alpha_{c_2}-2},$$

which only uses k hyper-parameters to define a prior over the k^2 values. Derive the MAP estimate under this prior (assuming we use k^2 variables to parameterize Θ)

3. We often use discrete distributions as parts of more-complicated distributions (like mixture models and graphical models). In these cases we often need to fit a weighted NLL for the form

$$f(\Theta) = -\sum_{i=1}^{n} v_i \log p(x_1^i, x_2^i \mid \Theta),$$

with n non-negative weights. What is the MLE for the k^2 elements of Θ under this weighting.

4. Consider the following alternate parameterization of the joint distribution: we use θ_{c_1} as the ("marginal") probability that variable 1 is in state c_1 ($p(x_1 = c_1) = \theta_{c_1}$) and $\theta_{c_2 \mid c_1}$ as the ("conditional") probability that variable 2 is in state c_2 conditioned on variable 1 being in state c_1 ($p(x_2 = c_2 \mid x_1 = c_1) = \theta_{c_2 \mid c_1}$).[2] Give the MLE for the k elements of θ_{c_1} and k^2 elements of $\theta_{c_2 \mid c_1}$ (you don't need to give the full derivation, but should know how this would be done).

[2] So we could use $p(x_1, x_2) = p(x_2 \mid x_1)p(x_1)$ to get the joint probability.

Hint: it may be convenient to write the discrete likelihood for an example i in the form

$$p(x^i \mid \Theta) = \prod_{c \in [k]^2} \theta_c^{\mathcal{I}[x^i = c]},$$

where c is a vector containing (c_1, c_2), $[x^i = c]$ evaluates to 1 if all elements are equal, and $[k]^2$ is all ordered pairs (c_1, c_2). You can use a Lagrangian to enforce the sum-to-1 constraint on the log-likelihood (bug the TAs to show you how to do this in office hours or tutorial if you don't know how), and you may find it convenient to define $n_c = \sum_{i=1}^{n} \mathcal{I}[x^i = c]$. Note that the solutions you find using the Lagrangian will already satisfy the non-negative constraint on the parameters.

2.2 Gaussian Self-Conjugacy

Consider n IID samples x^i distributed according to a Gaussian with mean μ and covariance $\sigma^2 I$,

$$x^i \sim \mathcal{N}(\mu, \sigma^2 I).$$

Assume that μ itself is distributed according to a Gaussian

$$\mu \sim \mathcal{N}(\mu_0, \Sigma_0),$$

with mean μ_0 and (positive-definite) covariance Σ_0. In this setting, the posterior for μ also follows a Gaussian distribution.[3]

Derive the form of the posterior distribution, $p(\mu \mid X, \sigma^2, \mu_0, \Sigma_0)$.
Hint: the posterior is a product of Gaussian densities.

[3]Because of the prior and posterior coming from the same family, we say that the Gaussian distribution is the "conjugate prior" for the Gaussian mean parameter (we'll formally discuss conjugate priors later in the course). Another reason the Gaussian distribution is important is that is the only (non-trivial) continuous distribution that has this "self-conjugacy" property.

2.3 Generative Classifiers with Gaussian Assumption

Consider the 3-class classification dataset in this image: In this dataset, we have 2 features and each colour represents one of the classes. Note that the classes are highly-structured: the colours each roughly follow a Gausian distribution plus some noisy samples.

Since we have an idea of what the features look like for each class, we might consider classifying inputs x using a *generative classifier*. In particular, we are going to use Bayes rule to write

$$p(y^i = c \mid x^i, \Theta) = \frac{p(x^i \mid y^i = c, \Theta) \cdot p(y^i = c \mid \Theta)}{p(x^i \mid \Theta)},$$

where Θ represents the parameters of our model. To classify a new example $\tilde{x}^i$, generative classifiers would use

$$\hat{y}^i = \underset{y \in \{1,2,\dots,k\}}{\arg\max}\ p(\tilde{x}^i \mid y^i = c, \Theta) p(y^i = c \mid \Theta),$$

where in our case the total number of classes k is 3.[4] Modeling $p(y^i = c \mid \Theta)$ is easy: we can just use a k-state categorical distribution,

$$p(y^i = c \mid \Theta) = \theta_c,$$

where θ_c is a single parameter for class c. The maximum likelihood estimate of θ_c is given by n_c/n, the number of times we have $y^i = c$ (which we've called n_c) divided by the total number of data points n.

Modeling $p(x^i \mid y^i = c, \Theta)$ is the hard part: we need to know the *probability of seeing the feature vector x^i given that we are in class c*. This corresponds to solving a density estimation problem for each of the k possible classes. To make the density estimation problem tractable, we'll assume that the distribution of x^i given that $y^i = c$ is given by a $\mathcal{N}(\mu_c, \Sigma_c)$ Gaussian distribution for a class-specific μ_c and Σ_c,

$$p(x^i \mid y^i = c, \Theta) = \frac{1}{(2\pi)^{\frac{d}{2}} |\Sigma_c|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(x^i - \mu_c)^T \Sigma_c^{-1} (x^i - \mu_c)\right).$$

Since we are distinguishing between the probability under k different Gaussians to make our classification, this is called *Gaussian discriminant analysis* (GDA). In the special case where we have a constant $\Sigma_c = \Sigma$ across all classes it is known as *linear discriminant analysis* (LDA) since it leads to a linear classifier between any two classes (while the region of space assigned to each class forms a convex polyhedron as in k-means clustering and softmax classification). Another common restriction on the Σ_c is that they are diagonal matrices, since this only requires $O(d)$ parameters instead of $O(d^2)$ (corresponding to assuming that the features are independent univariate Gaussians given the class label). Given a dataset $\mathcal{D} = \{(x^i, y^i)\}_{i=1}^n$, where $x^i \in \mathbb{R}^d$ and $y^i \in \{1, \dots, k\}$, the maximum likelihood estimate (MLE) for the μ_c and Σ_c in the GDA model is the solution to

$$\underset{\mu_1, \mu_2, \dots, \mu_k, \Sigma_1, \Sigma_2, \dots, \Sigma_k}{\arg\max} \prod_{i=1}^n p(x^i \mid y^i, \mu_{y^i}, \Sigma_{y^i}).$$

This means that the negative log-likelihood will be equal to

$$\begin{aligned} -\log p(X \mid y, \Theta) &= -\sum_{i=1}^n \log p(x^i, y^i \mid \mu_{y^i}, \Sigma_{y^i}) \\ &= \sum_{i=1}^n \frac{1}{2}(x^i - \mu_{y^i})^T \Sigma_{y^i}^{-1} (x^i - \mu_{y^i}) + \frac{1}{2}\sum_{i=1}^n \log |\Sigma_{y^i}| + \text{const.} \end{aligned}$$

In class we derived the MLE for this model under the assumption that we use full covariance matrices and that each class has its own covariance.

[4] The denominator $p(\tilde{x}^i \mid \Theta)$ is irrelevant to the classification since it is the same for all y.

1. Derive the MLE for the GDA model under the assumption of *common diagonal covariance* matrices, $\Sigma_c = D$ (d parameters). (Each class will have its own mean μ_c.)
2. Derive the MLE for the GDA model under the assumption of *individual scaled-identity* matrices, $\Sigma_c = \sigma_c^2 I$ (k parameters).
3. When you run *example_generative* it loads a variant of the dataset in the figure that has 12 features and 10 classes. This data has been split up into a training and test set, and the code fits a k-nearest neighbour classifier to the training set then reports the accuracy on the test data (around $\sim 63\%$ test error). The k-nearest neighbour model does poorly here since it doesn't take into account the Gaussian-like structure in feature space for each class label. Write a function *gda* that fits a GDA model to this dataset (using individual full covariance matrices). Hand in the function and report the test set accuracy.
4. In this question we would like to replace the Gaussian distribution of the previous problem with the more robust multivariate-t distribution so that it isn't influenced as much by the noisy data. Unlike the previous case, we don't have a closed-form solution for the parameters. However, if you run *example_student* it generates random noisy data and fits a multivariate-t model. By using the *studentT* model, write a new function *tda* that implements a generative model that is based on the multivariate-t distribution instead of the Gaussian distribution. Report the test accuracy with this model.

Hints: you may be able to substantially simplify the notation in the MLE derivations if you use the notation $\sum_{i \in y_c}$ to mean the sum over all values i where $y^i = c$. Similarly, you can use n_c to denote the number of cases where $y_i = c$, so that we have $\sum_{i \in y_c} 1 = n_c$. Note that the determinant of a diagonal matrix is the product of the diagonal entries, and the inverse of a diagonal matrix is a diagonal matrix with the reciprocals of the original matrix along the diagonal.

For the implementation you can use the result from class regarding the MLE of a general multivariate Gaussian. At test time for GDA, you may find it more numerically reasonable to compare log probabilities rather than probabilities of different classes, and you may find it helpful to use the *logdet* function to compute the log-determinant in a more numerically-stable way than taking the log of the determinant. (Also, don't forget to center at training and test time.)

For the last question, you may find it helpful to define an empty array that can be filled with k *DensityModel* objects using:

```
subModel = Array{DensityModel}(undef,k)
```

3 Mixture Models and Expectation Maximization

3.1 Categorical Mixture Model

Consider density estimation with examples $x^i \in \{1, 2, \ldots, k\}^d$ representing a set of d categorical variables. In this setting, a natural way to model an individual variable x_j^i would be with a categorical distribution,

$$p(x_j^i = c \mid \theta_{j,1}, \theta_{j,2}, \ldots, \theta_{j,k}) = \theta_{j,c},$$

so the joint distribution would be

$$p(x^i \mid \theta_{j,1}, \theta_{j,2}, \ldots, \theta_{j,k}) = \prod_{j=1}^{d} \theta_{j,x_j^i},$$

where all $\theta_{j,c} \geq 0$ and $\sum_{c=1}^{k} \theta_{j,c} = 1$ for each feature j. However, in this distribution the variables would are independent. One way to model dependent count variables would be with a mixture of m independent categorical distributions,

$$p(x^i \mid \Theta) = \sum_{c=1}^{m} p(z^i = c \mid \Theta^t) p(x^i \mid z^i = c, \Theta^t) = \sum_{c=1}^{m} \pi_c \prod_{j=1}^{d} \theta_{j,x_j^i}^c,$$

where:

- π_c is the probability that the examples comes from mixture c, $p(z^i = c \mid \Theta) = \pi_c$.
- $\theta_{j,c'}^c$ is the probability that x_j^i is c' for examples from mixture c, $p(x_j^i = c' \mid z^i = c, \Theta)$.
- We use Θ as the set containing all the π_c and $\theta_{j,c'}^c$ values.

Derive the EM update for this mixture model (treating the z^i as missing values).

Hint: a lot of the work has been done for you in the EM notes on the course webpage.

3.2 Semi-Supervised Gaussian Discriminant Analysis

Consider fitting a GDA model where some of the y^i values are missing at random. In particular, let's assume we have a set of n labeled examples (x^i, y^i) and another set of t unlabeled examples (x^i). This is a special case of *semi-supervised learning*, and fitting generative models with EM is one of the oldest semi-supervised learning techniques. When the classes exhibit clear structure in the feature space, it can be very effective even if the number of labeled examples is very small.

1. Derive the EM update for fitting the parameters of a GDA model (with individual full covariance matrices) in the semi-superivsed setting where we have n labeled examples and t unlabeled examples.

2. If you run the demo *example_SSL.jl*, it will load a variant of the dataset from the GDA, but where the number of labeled examples is small and a large number of unlabeled examples are available. The demo first fits a KNN model and then a generative Gaussian model (once you are finished the GDA question and assuming that you put your *gda* function in a file named *gda.jl*). Because the number of labeled examples it quite small, the performance is worse than in the GDA question. Write a function *gdaSSL* that fits the generative Gaussian model of the previous question using EM to incorporate the unlabeled data. Hand in the function and report the test error when training on the full dataset.

3. Repeat the previous part, but using the imputation approach ("hard"-EM) where we explicitly classify all the unlabeled examples before each model update. How does this change the performance and the number of iterations?

Hint: for the first question most of the work has been done for you in the EM notes on the course webpage. You can use the result (**) and the update of θ_c from those notes, but you will need to work out the update of the parameters of the Gaussian distribution $p(x^i|y^i, \Theta)$.

For the second question, although EM often leads to simple updates, implementing them correctly can often be a pain. One way to help debug your code is to compute the observed-data log-likelihood after every iteration. If this number goes down, then you know your implementation has a problem. You can also test your updates of sets of variables in this way too. For example, if you hold the μ_c and Σ_c fixed and only update the θ_c, then the log-liklihood should not go down. In this way, you can test each combination of updates on its own to make sure they are correct.

4 Very-Short Answer Questions

Give a short and concise 1-sentence answer to the below questions.

1. What is a situation where we can violate the golden rule on our test set, and still have it be a reasonable approximation of test error?
2. Suppose we have a way to generate new IID samples for our problem. If we are fitting a model with SGD, what would be the key advantage of using new IID samples as opposed to sampling from a fixed training set?
3. Is the minimum of a convex function achieved by a unique input value? What about a strictly-convex function or a strongly-convex function?
4. How can we use density estimation for supervised learning?
5. How many parameters does the general discrete distribution have when each feature is a categorical variable that can take up to k values.

6. What is the relationship between the covariance matrix Σ and the precision matrix Θ of a multivariate Gaussian?
7. Suppose we run the graphical LASSO method and it returns a tri-diagonal precision matrix. What would the graph look like?
8. Suppose we have a lot of extreme outliers in our dataset. Why is this less of a problem for a mixture of Gaussians than if we use a single Gaussian?
9. Why do the π_c need to be a convex combination in mixture models?

10. What is the relationship between the supervised naive Bayes model and the unsupervised mixture of Bernoullis model?
11. What is the difference between "missing at random" and "missing completely at random"?
12. What is an advantage of the Epanechnikov kernel over the Gaussian kernel.
13. How does parameter tieing allow us to model training examples that have different sizes?

CPSC 540 Project Proposal

This section is only to be done by students enrolled in CPSC 540.

For the final part of this assignment, you must a submit a project proposal for your course project. The proposal should be a maximum of 2 pages (and 1 page or half of a page is ok if you can describe your plan concisely). The proposal should be written for me and the TAs, so you don't need to introduce any ML background but you will need to introduce non-ML topics. The projects must be done in groups of 2-3, and will be submitted separately from the assignment.

There is quite a bit of flexibility in terms of the type of project you do, as I believe there are many ways that people can make valuable contributions to research. However, note that ultimately the project will have three parts:

1. A very short paper review summarizing the pros and cons of a particular paper on the topic (due with Assignment 3).
2. A short literature review summarizing at least 10 papers on a particular topic (due with Assignment 4).
3. A final report containing at most 6 pages of text (the actual document can be longer due to figures, tables, references, and proofs) that emphasizes a particular "contribution" (i.e., what doing the project has added to the world).

The three mains ingredients of the project proposal are:

1. What problem you are focusing on.
2. What you plan to do.
3. What will be the "contribution".

Also, note that for the course project that negative results (like "we tried something that we thought we would work in a particular setting but it didn't work") are acceptable (and often unavoidable).

Here are some standard project "templates" that you might want to follow:

- **Application bake-off**: you pick a specific application (from your research, personal interests, or maybe from Kaggle) or a small number of related applications, and try out a bunch of techniques (e.g., random forests vs. logistic regression vs. generative models). In this case, the contribution would be showing that some methods work better than others for this specific application (or your contribution could be that everything works equally well/badly).
- **New application**: you pick an application where where people aren't using ML, and you test out whether ML methods are effective for the task. In this case, the contribution would be knowing whether ML is suitable for the task.
- **Scaling up**: you pick a specific machine learning technique, and you try to figure out how to make it run faster or on larger datasets (for example, how do we apply kernel methods when n is very large). In this case, the contribution would be the new technique and an evaluation of its performance, or could be a comparison of different ways to address the problem.
- **Improving performance**: you pick a specific machine learning technique, and try to extend it in some way to improve its performance (for example, how can we efficiently use non-linearity within graphical models). In this case, the contribution would be the new technique and an evaluation of its performance.
- **Generalization to new setting**: you pick a specific machine learning technique, and try to extend it to a new setting (for example, making a graphical-model version of random forests). In this case,

the contribution would be the new technique and an evaluation of its performance, or could be a comparison of different ways to address the problem.

- **Perspective paper**: you pick a specific topic in ML, read a larger number of papers on the topic, then write a report summarizing what has been done on the topic and what are the most promising directions of future work. In this case, the contribution would be your summary of the relationships between the existing works, and your insights about where the field is going.
- **Coding project**: you pick a specific method or set of methods (like independent component analysis), and build an implementation of them. In this case, the contribution could be the implementation itself or a comparison of different ways to solve the problem.
- **Theory**: you pick a theoretical topic (like the variance of cross-validation or the convergence of proximal stochastic gradient in the non-convex setting), read what has been done about it, and try to prove a new result (usually by relaxing existing assumptions or adding new assumptions). The contribution could be a new analysis of an existing method, or why some approaches to analyzing the method will not work.

The above are just suggestions, and many projects will mix several of these templates together, but if you are having trouble getting going then it's best to stick with one of the above templates. Also note that the above includes topics not covered in the course (like random forests), so there is flexibility in the topic, but the topic should be closely-related to ML.

This question is mandatory but will not be formally marked: it's just a sanity check that you have at least one project idea that fits within the scope of a 540 course project, and it's an excuse for you to allocate some time to thinking about the project. Also, there is flexibility in the choice of project topics even after the proposal: if you want to explore different topics you can ultimately choose to do a project that is unrelated to the one in your proposal/paper-review/literature-review, although it will likely be easier to do all 4 parts on the same topic.