CPSC 540 Assignment 3 (due February 27)

Density Estimation and Project Proposal

1 Discrete and Gaussian Variables

1.1 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

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

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. Because of the sum-to-1 constraint, there are only (k^2-1) degrees of freedom in the discrete distribution, and not k^2. Derive the MLE for this distribution assuming that

$$\theta_{k,k} = 1 - \sum_{c_1=1}^{k}\sum_{c_2=1}^{k}\mathcal{I}[c_1 \neq k, c_2 \neq k]\theta_{c_1,c_2},$$

   so that the distribution only has $(k^2 - 1)$ parameters.
3. If we had separate parameter θ_{c_1} and θ_{c_2} for each variables, a reasonable choice of a prior would be a product of Dirichlet distributions,

$$p(\theta_{c_1}, \theta_{c_2}) \propto \theta_{c_1}^{\alpha_{c_1}-1}\theta_{c_2}^{\alpha_{c_2}-1}.$$

   For the general discrete distribution, a prior encoding the same assumptions would be

$$p(\theta_{c_1,c_2}) \propto \theta_{c_1,c_2}^{\alpha_{c_1}+\alpha_{c_2}-2}.$$

   Derive the MAP estimate under this prior.

Hint: it is convenient to write the likelihood for an example i in the form

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

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 the Lagrangian to enforce the sum-to-1 constraint on the log-likelihood, and you may find it convenient to define $N_c = \sum_{i=1}^{n} \mathcal{I}[x^i = c]$.

Solution

1.
Having n training examples $\mathbf{x} := (x^1, \ldots, x^n)$ where $x^i \in \{1, 2, \ldots, k\}^2$ with

$$\mathbb{P}(x_1^i = c_1, x_2^i = c_2) = \theta_{c1,c2}.$$

By denoting Θ the matrix containing all the parameters θ_{c_1,c_2}, we can write the likelihood for the i-th variable as

$$\mathbb{P}(x^i|\Theta) = \prod_{c\in[k]^2} \theta_c^{\mathcal{I}[x^i=c]}.$$

By assuming independence between the variables we conclude

$$\begin{aligned}\mathbb{P}(\mathbf{x}|\Theta) &= \prod_{i=1}^{n} \mathbb{P}(x^i|\Theta) \\ &= \prod_{i=1}^{n} \prod_{c\in[k]^2} \theta_c^{\mathcal{I}[x^i=c]}.\end{aligned}$$

Exchanging the order in the products and defining $N_c = \sum_{i=1}^{n} \mathcal{I}[x^i = c]$, we get

$$\mathbb{P}(\mathbf{x}|\Theta) = \prod_{c\in[k]^2} \theta_c^{N_c}. \tag{1}$$

Since the logarithm is an increasing function, the maximizer of this equation is the same as the maximizer of

$$\log(\mathbb{P}(\mathbf{x}|\Theta)) = \sum_{c\in[k]^2} N_c \log(\theta_c). \tag{2}$$

Since we want the parameters θ_c to represent probabilities, we have the constraint $\sum_{c\in[k]^2} \theta_c = 1$. Hence if we use Lagrange multipliers we conclude that for all $l \in [k]$ we have

$$\frac{\partial}{\partial\theta_l}(\log(\mathbb{P}(\mathbf{x}|\Theta))) = \lambda \frac{\partial}{\partial\theta_l}(\sum_{c\in[k]^2} \theta_c).$$

Derivating and solving for θ_l we conclude that

$$\theta_l = \frac{N_l}{\lambda}.$$

Inserting this result into the constraint we get

$$\lambda = \sum_{c\in[k]^2} N_c.$$

Hence the maximum likelihood estimation gives for each $l \in [k]^2$

$$\theta_l = \frac{N_l}{\sum_{c\in[k]^2} N_c}.$$

2.
In this case by assuming $\theta_{k,k} = 1 - \sum_{c \neq (k,k)} \theta_c$ we modify equation (1) to get

$$\mathbb{P}(\mathbf{x}|\Theta) = (1 - \sum_{c \neq (k,k)} \theta_c)^{\alpha} \prod_{c \neq (k,k)} \theta_c^{N_c},$$

where $\alpha = n - \sum_{c \neq (k,k)} N_c$. By taking log on both sides of the equation, we obtain

$$\log(\mathbb{P}(\mathbf{x}|\Theta)) = \alpha \log(1 - \sum_{c \neq (k,k)} \theta_c) + \sum_{c \neq (k,k)} N_c \log(\theta_c).$$

By letting $c = (c1, c2)$ and taking the derivative with respect to θ_{mn} we get

$$\frac{\partial}{\partial \theta_{mn}}(\log(\mathbb{P}(\mathbf{x}|\Theta))) = \frac{-\alpha \delta_{(m,n)(c_1,c_2)} \theta_{c_1,c_2}}{1 - \sum_{(c_1,c_2) \neq (k,k)} \theta_{c_1,c_2}} + \sum_{(c_1,c_2) \neq (k,k)} \frac{N_c}{\theta_{c_1,c_2}} \delta_{(m,n)(c_1,c_2)}.$$

Simplifying and equating to zero we obtain

$$\frac{\alpha \theta_{m,n}}{1 - \sum_{c \neq (k,k)} \theta_c} - \frac{N_{m,n}}{\theta_{m,n}} = 0.$$

If we define $A_{mn} = \sum_{c \neq (k,k), c \neq (m,n)}$ then we get

$$\alpha \theta_{mn}^2 + N_{mn} \theta_{mn} + N_{mn}(A_{mn} - 1) = 0.$$

With this we have $n \times m$ equations to solve for the $n \times m$ parameters. The solution gives us the vales of Θ for the MLE. Clearly is easier to use Lagrange Multipliers.

3.
We want to find the MAP for $\mathbb{P}(\Theta|\mathbf{x})$ or $\log(\mathbb{P}(\Theta|\mathbf{x}))$. Using Bayes rule we get

$$\mathbb{P}(\Theta|\mathbf{x}) \propto \mathbb{P}(\mathbf{x}|\Theta)\mathbb{P}(\Theta).$$

Choosing the prior

$$\mathbb{P}(\Theta) = \prod_{(c_1,c_2)\in[k]^2} \theta_{c1,c2}^{\alpha_{c_1}+\alpha_{c_2}-2}.$$

We can easily see that

$$\log(\mathbb{P}(\Theta)) = \sum_{(c_1,c_2)\in[k]^2} (\alpha_{c_1}+\alpha_{c_2}-2)\log(\theta_{c_1,c_2}).$$

Combining this result with equation (2) we get that the log posterior is given by

$$\log(\mathbb{P}(\Theta|\mathbf{x})) = \underbrace{\sum_{(c_1,c_2)\in[k]^2} N_{c_1,c_2}\log(\theta_{c_1,c_2})}_{\log(\mathbb{P}(\mathbf{x}|\Theta))} + \underbrace{\sum_{(c_1,c_2)\in[k]^2} (\alpha_{c_1}+\alpha_{c_2}-2)\log(\theta_{c_1,c_2})}_{\log(\mathbb{P}(\Theta))}.$$

Using the sum to one constraint and Lagrange multipliers we conclude that for all $(l_1,l_2)\in[k]^2$ we must have

$$\frac{\partial}{\partial\theta_{l_1,l_2}}\Big(\sum_{(c_1,c_2)\in[k]^2} N_{c_1,c_2}\log(\theta_{c_1,c_2}) + \sum_{(c_1,c_2)\in[k]^2} (\alpha_{c_1}+\alpha_{c_2}-2)\log(\theta_{c_1,c_2})\Big) = \lambda\frac{\partial}{\partial\theta_{l_1,l_2}}\Big(\sum_{(c_1,c_2)\in[k]^2}\theta_{c_1,c_2}\Big)$$

Taking the derivatives and solving for θ_{l_1,l_2} we get

$$\theta_{l1,l2} = \frac{N_{l1,l2}+\alpha_{l_1}+\alpha_{l_2}-2}{\lambda}.$$

Plugging in this value into the sum to one constraint we conclude

$$\lambda = \sum_{(c_1,c_2)\in[k]^2} N_{c1,c2}+\alpha_{c_1}+\alpha_{c_2}-2.$$

Hence the MAP estimate is

$$\theta_{l1,l2} = \frac{N_{l1,l2}+\alpha_{l_1}+\alpha_{l_2}-2}{\sum_{(c_1,c_2)\in[k]^2} N_{c1,c2}+\alpha_{c_1}+\alpha_{c_2}-2}.$$

1.2 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

$$p(y = c|x, \Theta) = \frac{p(x|y = c, \Theta) \cdot p(y = c|\Theta)}{p(x|\Theta)},$$

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

$$\hat{y} = \underset{y \in \{1,2,\ldots,k\}}{\arg\max}\ p(\hat{x}|y = c, \Theta)p(y = c|\Theta),$$

where in our case the total number of classes k is 3 (The denominator $p(\hat{x}|\Theta)$ is irrelevant to the classification since it is the same for all y.) Modeling $p(y = c|\Theta)$ is eays: we can just use a k-state categorical distribution,

$$p(y = c|\Theta) = \theta_c,$$

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 $p(x|y = c, \Theta)$ is the hard part: we need to know the *probability of seeing the feature vector x 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 given that $y = c$ is given by a $\mathcal{N}(\mu_c, \Sigma_c)$ Gaussian distribution for a class-specific μ_c and Σ_c,

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

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). 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 $y^i \in \{1, \ldots, k\}$, the maximum likelihood estimate (MLE) for the μ_c and Σ_c in the GDA model is the solution to

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

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

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

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 scale-identity* matrices, $\Sigma_c = \sigma_c^2 I$ (k parameters).

3. It's painful to derive these from scratch, but you should be able to see a pattern that would allow other common restrictions. Without deriving the result from scratch (hopefully), give the MLE for the case of *individual full* covariance matrices, Σ_c ($O(kd^2)$ parameters).

4. 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 ($\sim 36\%$). 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 *generativeGaussian* that fits a GDA model to this dataset (using individual full covariance matrices). Hand in the function and report the test set accuracy.

5. 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_tdist* it generates random noisy data and fits a multivariate-t model (you will need to add the *minFunc* directory to the Matlab path for the demo to work). By using the *multivariateT* model, write a new function *generativeStudent* 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 will be able to substantially simplify the notation in parts 1-3 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 part three you can use the result from class regarding the MLE of a general multivariate Gaussian. You may find it helpful to use the included *logdet.m* function to compute the log-determinant in more numerically-stable way.

Solution

1.
Under the assumption that $\Sigma_c = D$ where D diagonal, the negative log-likelihood becomes

$$-\log \mathbb{P}(X|y,\Theta) = \sum_{i=1}^{n} \frac{1}{2}(x^i - \mu_{y^i})^T D^{-1}(x^i - \mu_{y^i}) + \frac{1}{2}\sum_{i=1}^{n} \log|D| + \text{const.}$$

This is a quadratic function in the μ_{y^i} variables, hence it has a unique minimum. By taking the gradient with respect to μ_c and equating it to zero we get

$$\nabla_{\mu_c}(-log(\mathbb{P}(X|y,\theta))) = \sum_{i \in y_c} D^{-1}(\mu_c - x^i) = 0,$$

in this last equation we used the identity $\nabla_z(z^T A z) = (A + A^T)z$, and the fact that D^{-1} is symmetric. Solving for μ_c we get

$$\mu_c \sum_{i \in y_c} 1 = D \sum_{i \in y_c} D^{-1} x^i.$$

If we denote $N_c = \sum_{i \in y_c} 1$ and multiply D with D^{-1} we conclude that

$$\mu_c = \frac{1}{N_c} \sum_{i \in y_c} x^i.$$

That is, the maximum likelihood estimate for the mean is the the average over all features that correspond to the class c.

To calculate the MLE estimate for D, let us assume that $D = diag(\frac{1}{\lambda_1}, \ldots, \frac{1}{\lambda_d})$, hence $D^{-1} = diag(\lambda_1, \ldots, \lambda_d)$. Hence we have

$$-\log \mathbb{P}(X|y,\Theta) = \sum_{i=1}^{n} \frac{1}{2} \sum_{j=1}^{d} \lambda_j (x^i - \mu_{y^i})_j^2 + \frac{1}{2} \sum_{i=1}^{n} \log(\prod_{j=1}^{d} \frac{1}{\lambda_j}) + const.$$

In this equation the subscript j in $(x^i - \mu_{y^i})_j$ means the j-th component of the vector. Using the properties of the logarithm we get

$$-\log \mathbb{P}(X|y,\Theta) = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{d} \lambda_j (x^i - \mu_{y^i})_j^2 - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{d} \log(\lambda_j) + const.$$

Taking the partial derivative with respect to the l-th element and equating to 0 we get

$$\frac{\partial}{\partial \lambda_l}(-\log \mathbb{P}(X|y,\Theta)) = \frac{1}{2} \sum_{i=1}^{n} (x^i - \mu_{y^i})_l^2 - \frac{n}{2} \frac{1}{\lambda_l} = 0.$$

Solving for λ_l we conclude

$$\lambda_l = \frac{1}{n} \sum_{i=1}^{n} (x^i - \mu_{y^i})_l^2.$$

This means that the best estimate for the l-th element in the diagonal of D^{-1} is the estimate of the variance along the l-th coordinate direction.

2.
Now we are going to assume that $\Sigma_c = \sigma_c I$. With this assumption the negative log-likelihood becomes

$$-\log \mathbb{P}(X|y,\Theta) = \sum_{i=1}^{n} \frac{1}{2\sigma_{y^i}^2} \underbrace{(x^i - \mu_{y^i})^T (x^i - \mu_{y^i})}_{\|x^i - \mu_{y^i}\|_2^2} + \frac{1}{2} \sum_{i=1}^{n} \log |\sigma_{y^i}^2 I| + \text{const.}$$

Again this is a quadratic function in terms of the means, hence by finding the gradient and equating to zero we find the unique minimizer. For the c class label we have

$$\nabla_{\mu_c}(-log(\mathbb{P}(X|y,\theta))) = \sum_{i \in y_c} \frac{1}{\sigma_c^2}(\mu_c - x^i) = 0.$$

Solving for the mean we get

$$\mu_c = \frac{1}{N_c} \sum_{i \in y_c} x^i. \tag{3}$$

Now we are going to find the MLE for the σ_c. To do so we find the partial derivative with respect to σ_l of the negative log-likelihood to get

$$\frac{\partial}{\partial \sigma_l}(-\log \mathbb{P}(X|y,\Theta)) = -\frac{1}{\sigma_l^3}\|x^l - \mu_l\|^2 + \frac{d}{\sigma_l} = 0.$$

Solving for σ^2 we conclude

$$\sigma_l^2 = \frac{\|x^l - \mu_l\|^2}{d}. \tag{4}$$

This is the MLE estimate for the variances.

3.
Finally for the general case, the likelihood is

$$-\log(\mathbb{P}(X|y,\Theta) = \sum_{i=1}^{n} \frac{1}{2}(x^i - \mu_{y^i})^T \Sigma_{y^i}^{-1}(x^i - \mu_{y^i}) + \frac{1}{2}\sum_{i=1}^{n} \log|\Sigma_{y^i}| + \text{const.}$$

Again, taking the derivative with respect to μ_c and equate to 0 we get

$$\nabla_{\mu_c}(-\log(\mathbb{P}(X|y,\Theta)) = \sum_{i \in y_c} \Sigma_c^{-1}(\mu_c - x^i) = 0.$$

Solving for μ_c we get the same result as in the two previous cases, namely

$$\mu_c = \frac{1}{N_c}\sum_{i \in y_c} x^i.$$

For the covariance, things are a little bit more tricky. First recall that given a matrix A we have

$$\nabla_A \log|A| = A^{-T}. \tag{5}$$

On the other hand we have for matrices A and B

$$\nabla_A tr(BA) = B^T.$$

Since for any c we have that $(x^i - \mu_c)^T \Sigma_c^{-1}(x^i - \mu_c)$ is a number then

$$tr((x^i - \mu_c)^T \Sigma_c^{-1}(x^i - \mu_c)) = tr((x^i - \mu_c)(x^i - \mu_c)^T \Sigma_c^{-1}).$$

Hence

$$\nabla_{\Sigma_c^{-1}}(tr((x^i - \mu_c)(x^i - \mu_c)^T \sigma_c^{-1})) = (x^i - \mu_c)(x^i - \mu_c)^T.$$

Combining this result with equation (5) we conclude

$$\frac{\partial}{\partial \Sigma_c}(-\log(\mathbb{P}(X|y,\Theta))) = \frac{1}{2}\sum_{i \in y_c}(x^i - \mu_c)(x^i - \mu_c)^T - \frac{N_c}{2}\Sigma_c,$$

where in the last term we use the symmetry of Σ_c. Equating to 0 and solving for Σ_c we conclude

$$\Sigma_c = \frac{1}{N_c}\sum_{i \in y_c}(x^i - \mu_c)(x^i - \mu_c)^T.$$

This gives the MLE for the covariance matrix of the c class.

4.
Using the script generativeGaussian.m and running the script example_Generative.m we get the following output

```
>> example_generative
Gaussian Gen. Model. accuracy is 0.63
```

That is, we have an accuracy of 63% in the test set.
5.
Using the script generativeStudent.m and running the script example_Generative.m we get the following output

```
>> example_generative
Gaussian Gen. Model. accuracy is 0.63
Fitting multivariate student T density model...
Fitting multivariate student T density model...
Fitting multivariate student T density model...
Fitting multivariate student T density model...
Fitting multivariate student T density model...
Fitting multivariate student T density model...
Fitting multivariate student T density model...
Fitting multivariate student T density model...
Fitting multivariate student T density model...
Fitting multivariate student T density model...
Tdist Gen. Model. accuracy is 0.79
```

In this case the accuracy is improved and we get 79% of accuracy.

1.3 Self-Conjugacy for the Mean Parameter

If x is distributed according to a Gaussian with mean μ,

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

and we assume that μ itself is distributed according to a Gaussian

$$\mu \sim \mathcal{N}(\alpha, \gamma^2),$$

then the posterior $\mu|x$ also follows a Gaussian distribution.[1] Derive the form of the (Gaussian) distribution for $p(\mu|x, \alpha, \sigma^2, \gamma^2)$.

Hints: Use Bayes rule and use the $\propto$ sign to get rid of factors that don't depend on μ. You can "complete the square" to make the product look like a Gaussian distribution, e.g. when you have $\exp(ax^2 - bx + \text{const})$ you can factor out an a and add/subtract $(b/2a)^2$ to re-write it as

$$\begin{aligned}\exp\left(ax^2 - bx + const\right) &\propto \exp\left(ax^2 - bx\right) = \exp\left(a(x^2 - (b/a)x)\right) \\ &\propto \exp\left(a(x^2 - (b/a)x + (b/2a)^2)\right) = \exp\left(a(x - (b/2a))^2\right).\end{aligned}$$

Note that multiplying by factors that do not depend on μ within the exponent does not change the distribution. In this question you will want to complete the square to get the distribution on μ, rather than x. You may find it easier to solve thie problem if you parameterize the Gaussians in terms of their 'precision' parameters (e.g., $\lambda = 1/\sigma^2$, $\lambda_0 = 1/\gamma^2$) rather than their variances σ^2 and γ^2.

[1] 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.

Solution

By Bayes rule the posterior can be computed as

$$\mathbb{P}(\mu|x,\alpha,\sigma^2,\gamma^2) \propto \mathbb{P}(x|\mu,\alpha,\sigma^2,\gamma^2)\mathbb{P}(\mu|\alpha,\gamma^2).$$

By normality hypothesis on the likelihood and the prior we may write

$$\mathbb{P}(\mu|x,\alpha,\sigma^2,\gamma^2) \propto \exp(-\frac{\lambda}{2}(x-\mu)^2)\exp(-\frac{\lambda_0}{2}(\mu-\alpha)^2),$$

where $\lambda = \frac{1}{\sigma^2}$ and $\lambda_0 = \frac{1}{\gamma^2}$. Combining the exponentials we get

$$\mathbb{P}(\mu|x,\alpha,\sigma^2,\gamma^2) \propto \exp(-\frac{1}{2}(\lambda(x-\mu)^2 + \lambda_0(\mu-\alpha)^2)).$$

Expaning squares and keeping only terms depending on μ we obtain

$$\mathbb{P}(\mu|x,\alpha,\sigma^2,\gamma^2) \propto \exp(-\frac{1}{2}(\lambda\mu^2 - 2\mu\lambda x + \lambda_0\mu^2 - 2\mu\lambda_0\alpha)).$$

Rearranging this equation can be cast into

$$\mathbb{P}(\mu|x,\alpha,\sigma^2,\gamma^2) \propto \exp(-\frac{1}{2}((\lambda+\lambda_0)\mu^2 - 2\mu(\lambda x + \lambda_0\alpha))).$$

To complete squares we need to add and substract the term $\left(\frac{\lambda x+\lambda_0\alpha}{\lambda+\lambda_0}\right)^2$. Since adding this term is equivalent to add a constant term with respect to μ we may complete squares by just writing

$$\mathbb{P}(\mu|x,\alpha,\sigma^2,\gamma^2) \propto \exp(-\frac{1}{2}((\lambda+\lambda_0)\mu^2 - 2\mu(\lambda x + \lambda_0\alpha) + \left(\frac{\lambda x + \lambda_0\alpha}{\lambda+\lambda_0}\right)^2)).$$

Factoring the argument in the expoonential we get

$$\mathbb{P}(\mu|x,\alpha,\sigma^2,\gamma^2) \propto \exp(-\frac{1}{2}\left(\sqrt{\lambda+\lambda_0}\mu - \frac{\lambda x + \lambda_0\alpha}{\lambda+\lambda_0}\right)^2).$$

Factoring the term to the left of μ, this equation is written as

$$\mathbb{P}(\mu|x,\alpha,\sigma^2,\gamma^2) \propto \exp(-\frac{\lambda+\lambda_0}{2}\left(\mu - \frac{\lambda x + \lambda_0\alpha}{(\lambda+\lambda_0)^{\frac{3}{2}}}\right)^2).$$

This says that μ has also Gaussian distribution. More explicitly

$$\mu|x,\alpha,\sigma^2,\gamma^2 \sim \mathcal{N}(\frac{\lambda x + \lambda_0\alpha}{(\lambda+\lambda_0)^{\frac{3}{2}}}, \frac{1}{\lambda+\lambda_0}).$$

2 Mixture Models and Expectation Maximization

2.1 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 n labeled examples (x^i, y^i) and then another another 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 techqniues. 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*, it will load a variant of the dataset from the previous question, 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 Question 1). Because the number of labeled examples it quite small, the performance is worse than in Question 2. Write a function *generativeGaussianSSL* 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 "hard"-EM algorithm where we explicitly classify all the unlabeled examples. 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 $p(x^i|y^i, \Theta)$.

Hint: 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 of combinationso of updates on their to make sure they are correct.

Solution

1.1.
From equation (**) in class notes we know that

$$Q(\Theta|\Theta^t) = \sum_{i=1}^{n} \log(\mathbb{P}(y^i, x^i|\Theta)) + \sum_{i=1}^{t} \sum_{\tilde{y}_i} r^i_{\tilde{y}^i} \log(\mathbb{P}(\tilde{y}^i, \tilde{x}^i|\Theta)). \tag{6}$$

On the other hand using Bayes rule we have

$$\log(\mathbb{P}(y^i, x^i|\Theta)) = \log(\mathbb{P}(x^i|y^i, \Theta)) + \log(\mathbb{P}(y^i|\Theta)) + const.$$

By hypothesis $x^i|y^i, \Theta \sim \mathcal{N}(\mu_{y^i}, \Sigma_{y^i})$, and $\mathbb{P}(y^i|\Theta) = \frac{n_{y^i}}{n}$ where n_{y^i} is the number of y^i in the training sample. Hence we can write

$$\log(\mathbb{P}(y^i, x^i|\Theta)) = \frac{1}{2}(x^i - \mu_{y^i})^T \Sigma^{-1}_{y^i}(x^i - \mu_{y^i}) + \frac{1}{2}\log|\Sigma_{y^i}| + const. \tag{7}$$

If we apply Bayes rule to the argument in the second summand in equation (6) we get

$$\log(\mathbb{P}(\tilde{y}^i, \tilde{x}^i|\Theta)) = \log(\mathbb{P}(\tilde{x}^i|\tilde{y}^i, \Theta)) + \log(\mathbb{P}(\tilde{y}^i|\Theta)) + const.$$

As before $\tilde{x}^i|\tilde{y}^i, \Theta \sim \mathcal{N}(\mu_{\tilde{y}^i}, \Sigma_y)$ and by the class notes in EM we already know that $\mathbb{P}(\tilde{y}^i|\Theta) = \frac{\sum_{i=1}^t r^i_{\tilde{y}^i}}{t}$. Hence we can write

$$\log(\mathbb{P}(\tilde{y}^i, \tilde{x}^i|\Theta)) = \frac{1}{2}(x^i - \tilde{y}^i)^T \Sigma^{-1}_{\tilde{y}^i}(\tilde{x}^i - \mu_{\tilde{y}^i}) + \frac{1}{2}\log|\Sigma_{\tilde{y}^i}| + const. \tag{8}$$

Combining equations (7) and (8) we conclude

$$Q(\Theta|\Theta^t) = \sum_{i=1}^{n} \frac{1}{2}(x^i - \mu_{y^i})^T \Sigma^{-1}_{y^i}(x^i - \mu_{y^i}) + \frac{1}{2}\log|\Sigma_{y^i}| + \sum_{i=1}^{t} \sum_{\tilde{y}_i} r^i_y \left(\frac{1}{2}(\tilde{x}^i - \mu_{\tilde{y}^i})^T \Sigma^{-1}_{\tilde{y}^i}(\tilde{x}^i - \mu_{\tilde{y}^i}) + \frac{1}{2}\log|\Sigma_{\tilde{y}^i}|\right) + const.$$

To find the update Θ^{t+1} we optimize with respect to the means and covariance matrices. First it is straightforward to check that for a class c

$$\nabla_{\mu_c} Q(\Theta|\Theta^t) = -\sum_{i \in y_c} \Sigma_c^{-1}(x^i - \mu_c) - \sum_{i=1}^{t} r^i_c \Sigma_c^{-1}(\tilde{x}^i - \mu_c).$$

Equating to zero and solving for μ_c we find

$$\mu_c = \frac{\sum_{i \in y_c} x^i + \sum_{i=1}^t r^i_c \tilde{x}^i}{n_c + \sum_{i=1}^t r^i_c}. \tag{9}$$

Using the formulas for the gradient of the log-determinant and the trace as used in Question 1.2 we get

$$\nabla_{\Sigma_c^{-1}} Q(\Theta|\Theta^t) = \frac{1}{2}\sum_{i \in y_c}(x^i - \mu_c)(x^i - \mu_c)^T - \frac{n_c}{2}\Sigma_c + \frac{1}{2}\sum_{i=1}^{t} r^i_c(\tilde{x}^i - \mu_c)(\tilde{x}^i - \mu_c)^T - \frac{1}{2}\Sigma_c \sum_{i=1}^{t} r^i_c.$$

Equating to zero and solving for Σ_c we get

$$\Sigma_c = \frac{\sum_{i \in y_c}(x^i - \mu_c)(x^i - \mu_c)^T + \sum_{i=1}^t r^i_c(\tilde{x}^i - \mu_c)(\tilde{x}^i - \mu_c)^T}{n_c + \sum_{i=1}^t r^i_c}. \tag{10}$$

Finally the optimization for a given class c the optimal of θ_c is given by

$$\theta_c = \frac{n_c + \sum_{i=1}^{t} r_c^i}{n+t}. \tag{11}$$

By using the values in equations (9), (10) and (11) we obtain the update

$$\Theta^{t+1} = \arg\max_{\Theta} Q(\Theta|\Theta^t).$$

1.2.
By using the script generativeGaussianSSL.m and runing the script example_SSL.m we get the folowing output after 10 iterations in the EM algorithm

```
>> example_SSL
KNN accuracy is 0.28
Gaussian Gen. Model. accuracy is 0.49
SSL Gauss. Gen. Model. accuracy is 0.75
```

So we can see that for the semi-supervised learning setting, working with hidden variables gives better performance.

1.3
Using the script generativeGaussianSSLHard.m we get after 7 iterations the following result

```
>> example_SSL
SSL Gauss. Gen. Model. accuracy is 0.75
The time to run generativeGaussSSL is: Elapsed time is 6.322742 seconds.
SSL HARD Gauss. Gen. Model. accuracy is 0.75
The time to run generativeGaussHard is: Elapsed time is 3.704242 seconds.
```

So not only we saved 5 iterations by doing hard EM, we also reduced time of computation almost by half, to get the same accuracy.

2.2 Mixture of Bernoullis

The function *example_Bernoulli* loads a binarized version of the MNIST dataset and fits a density model that uses an independent Bernoulli to model each feature. If the reports the average NLL on the test data and shows 4 samples generated from the model. Unfortunately, the test NLL is infinity and the samples look terrible.

1. To address the problem that the average NLL is infinity, modify the *densityBernoulli* function to implement Laplace smoothing based on an extra argument α. Hand in the code and report the average NLL with $\alpha = 1$.

2. Write a new function implementing the mixture of Bernoullis model with Laplace smoothing of the θ values (note that Laplace smoothing only change the M-step). Hand in the code and report the average NLL with $\alpha = 1$ and $k = 10$ for a particular run of the algorithm, as well as 4 samples from the model and 4 of the cluster images.

Solutions:

2.1.
By modifying the script densityBernoulli.m to use Laplace smoothing ,i.e. $\alpha - 1 = 1$, we get the following output when we run the script example_Bernoulli.m

```
>> example_Bernoulli

averageNLL =

  205.8471
```

So with Laplace smoothing the NLL is now finite.

2.2
In this case we assume a model for the i-th sample of the form

$$\mathbb{P}(x^i|\Theta) = \sum_{c=1}^{10} \pi_c \mathbb{P}(x^i|\theta_c),$$

where

$$\Theta = (\theta, \pi).$$

Here θ is a 10×784 matrix that contains all the parameters in the Bernoulli model and π is a vector that contains $\pi_1, \pi_2, \ldots, \pi_{10}$. For each component we assume independence to get

$$\mathbb{P}(x^i|\theta_c) = \prod_{j=1}^{728} \mathbb{P}(x_j^i|\theta_{cj}),$$

and

$$\mathbb{P}(x_j^i|\theta_{cj}) = \theta_{cj}^{x_j^i}(1-\theta_{cj})^{1-x_j^i}.$$

Finally we assume 10 hidden variables with prior probabilities

$$\mathbb{P}(z^i = c|\Theta) = \pi_c.$$

With this we calculate the function $Q(\Theta|\Theta^t)$ and maximize it in order to find the update Θ^{t+1}. After doing some thedious algebra we get that the maximum of $Q(\Theta|\Theta^t)$ we need to choose that parameters as

$$\theta_{cl} = \frac{\sum_{i=1}^n r_l^i x_m^i}{\sum_{i=1}^n r_l^i},$$

and

$$\pi_c = \frac{\sum_{i=1}^n r_c^i + 1}{n+2},$$

where in the last line we used Laplace smoothing. With this update scheme we apply EM to optimize the likelihood $\mathbb{P}(x^i|\Theta)$ and apply it to the MNIST data set. The average negative log-likelihood was

```
mple_mixtureBernoulli

averageNLL =

  166.5713
```

Below are shown 4 samples from the model using the script mixtureBernoulli.m and example_mixtureBernoulli.m.

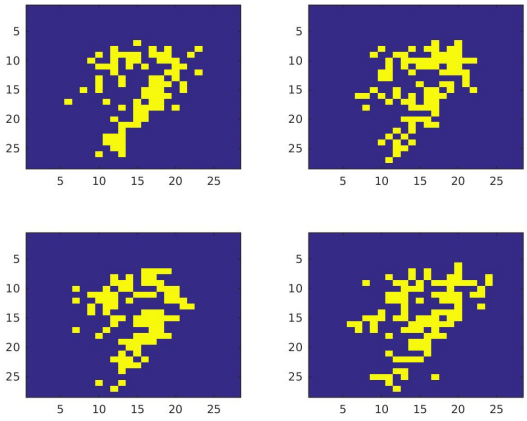

Figure 1: Four Samples from the Bernoulli Model

Below it is shown a sample for each one of the 10 clusters.

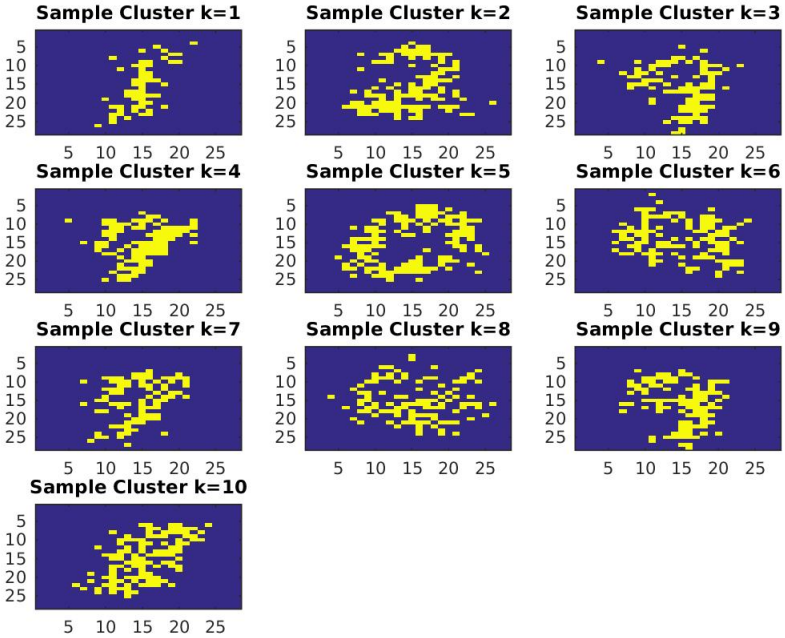

Figure 2: Samples for each of the ten clusters

Project Proposal

Applications of Machine Learning to Stock Market Predictions

Anson Wong, Gudbrand Duff Morris and Juan Garcia

Introduction

We propose to undertake an exploratory project on the potential impact of machine learning techniques to stock market prediction and analysis.

The idea of using algorithms to predict the price of assets is as old as the stock exchange itself, and the existing body of work on the subject is huge. Therefore, **the first part of our project will be an extensive literature review**. We will read and share interesting papers, using our findings to guide the development of the project scope and details.
Each one of the participants in this project have a possible idea in what direction this project could go:

Gudbrand's driving question: With the dawn of the Era of Big Data upon us, I seek to discover if and how a "change in magnitude will produce a change in kind".

Anson's asks: With the lack of insider information for amateur traders and the rise of algorithmic traders in the current stock market, is trading a lost cause for you and I? If not, then could an amateur machine learnist take a stab at trading and forecast price action with high success probability? I believe this is possible to earn large gains in niché markets using a healthy amount of machine learning and understanding of human psychology. We plan to explore reinforcement learning and feature selection to achieve such a goal.

Juan's Idea: A machine can perform better than a human in a wide range of situations. Forecasting and detecting patterns in the time series in the stock market could be one of these situations. My idea is that with the techniques and ideas we are learning, we can teach a machine how to recognize what a good investment is. The criteria to define 'good' comes from our knowledge from stock markets after doing some literature review. For the moment my first though in that direction is to apply the classification techniques we are learning, in order to select the portfolio with less variance. In this case the concept of 'good' investment is 'safe' investment.

Our literature research so far is summarized below

Literature Review

Papers:

- http://www-stat.wharton.upenn.edu/ steele/Courses/434/434Context/EfficientMarket/AndyLoJPM2004.pdf
- http://www-stat.wharton.upenn.edu/ steele/Courses/434/434Context/EfficientMarket/Granger-stockmarket.pdf

Interesting/useful links

- This guy is pretty great: `http://www-stat.wharton.upenn.edu/~steele/`
- `https://www.udacity.com/course/machine-learning-for-trading--ud501`

- https://www.quantopian.com/posts/simple-machine-learning-example
- http://www.qminitiative.org/UserFiles/files/S_Cl\%C3\%A9men\%C3\%A7on_ML.pdf
- http://www-stat.wharton.upenn.edu/~steele/Courses/9xx/Resources/MLFinancialApplications/MLFinance.html
- https://www.quora.com/How-do-financial-companies-use-machine-learning
- http://techemergence.com/machine-learning-in-finance-applications/
- Our GoogleDrive: https://drive.google.com/drive/folders/0B5_P6FZEZzQxOEN6ZTU1Q1czQTQ?usp=sharing
- R package for financial analysis: https://cran.r-project.org/web/views/Finance.html

Applications/Implementations/Ideas

As the famous Oscar Wilde quote goes; "A cynic is a man who knows the price of everything but the value of nothing". In this sense, a ML model can be regarded as the ultimate cynic. The cynic has both advantages and disadvantages compared with the expert investor. With this in mind we have a set of potential topics/ideas to start with:

Predictability How hard is the problem? Almost a philosophical question.

Clustering What sort of stocks/bonds/instruments are there? Which models work for different categories?

Feature Selection Which features are important when it comes to prediction? We see many examples of seemingly godly predictions made based on the most complex features. Is this sort of 'thinking' possible for machines? Take for example the fictional Silicon Valley investor Peter Gregory, who reasons from the popularity of Burger King to the impending surge in periodic cicadas to Indonesian sesame futures. Or hedge fund manager Micheal Burry, one of the few big shot investors to recognize the subprime mortgage crisis long before it happened, just by analyzing the right data.

Prediction techniques How can we predict the behaviour of a stochastic PDE (Black Scholes')?

Momentum Investing Gradient step based on today's winners. A potential winning strategy.

Page Rank Page rank is particularly useful for ranking connections (whether it be for search results, biological systems, social media influence). Although it might be a stretch, are there ways to use Page Rank to shed light on how the stock market works?

Q-learning A method of reinforcement learning which sets up a reward system, and uses a mix of exploration and exploitation to achieve optimization of a goal. Planning to apply this to create a trading monkey.

Financial Background The introduction should get the reader up to date on the subject. Efficient markets, price, information, volatility, portfolio, ...

Portfolio Selection Real-time optimization++

Finally we consider some technical aspects regarding data sets and software.

Other

- Datasets: There are plenty of sites were time series for the stock market can be accesed, such as, Finviz.com, Google Finance, Yahoo Finance, MarketWatch, to name a few.
- Programing Language and Packages: Any high level programing can be use to do the data analysis. At this point in the project there is some preference for R and the financial analysis packages such as Empirical Finance. This is since there is familiarity from the members of this research group with this language.