CPSC 340 Assignment 5

We are providing solutions because supervised learning is easier than unsupervised learning, and so we think having solutions available can help you learn. However, the solution file is meant for you alone and we do not give permission to share these solution files with anyone. Both distributing solution files to other people or using solution files provided to you by other people are considered academic misconduct. Please see UBC's policy on this topic if you are not familiar with it:
`http://www.calendar.ubc.ca/vancouver/index.cfm?tree=3,54,111,959`
`http://www.calendar.ubc.ca/vancouver/index.cfm?tree=3,54,111,960`

Important: Submission Format

Please make sure to follow the submission instructions posted on Piazza. **We are entitled to deduct 50% of your marks if the submission format is incorrect.**

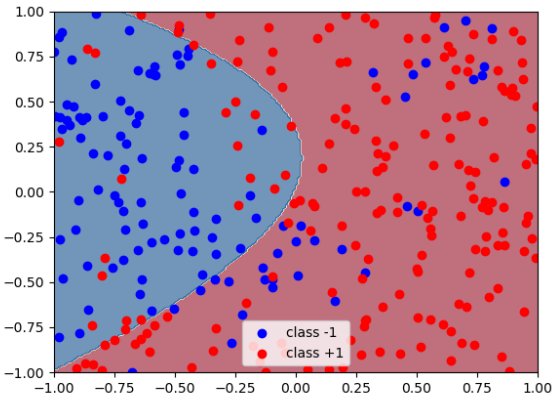
1 Kernel Logistic Regresion

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).

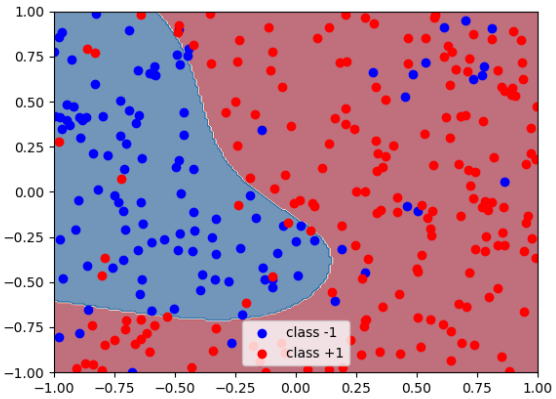
1.1 Implementing kernels

Inside `kernels.py`, you will see classes named `KernelPolynomial` and `KernelGaussianRBF`, whose `evaluate()` methods are yet to be written. 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 kernel hyperparameters $p = 2$ and $\sigma = 0.5$ respectively, and $\lambda = 0.01$ for the regularization strength.

Answer: For polynomial, the training error is 0.183 and the validation error is 0.17. Image below:



For RBF, we will accept two versions of the code, with or without the $\frac{1}{\sqrt{2\pi\sigma^2}}$ prefactor (they are equivalent, but change the scaling of the regularization, so will give different results for fixed λ). If this term is included, the training error is 0.127 and the validation error is 0.1. Image below:



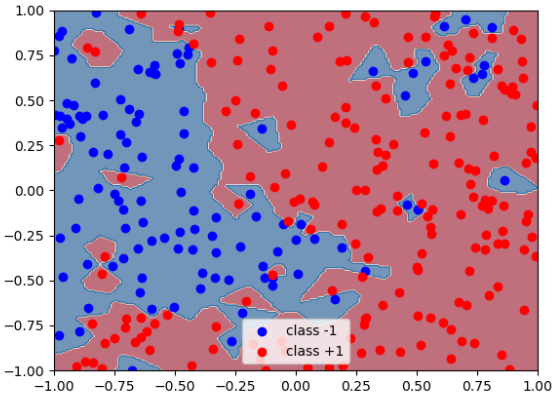
If this term is not included, the training error is also 0.127 but the validation error is 0.09. The figure looks almost exactly the same.

1.2 Hyperparameter search

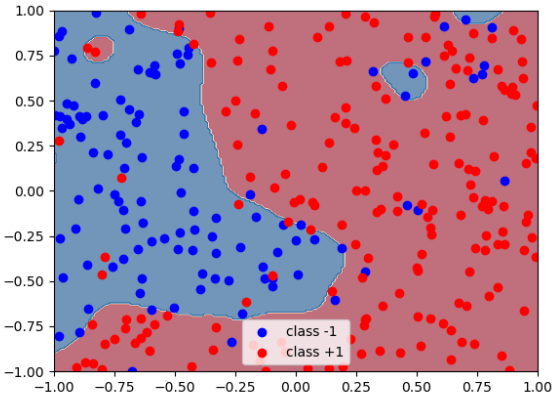
For the RBF kernel logistic regression, consider the hyperparameters values $\sigma = 10^m$ for $m = -2, -1, \ldots, 2$ and $\lambda = 10^m$ for $m = -4, -3, \ldots, 0$. In `main.py`, perform a grid search over the possible combinations of these hyperparameter values, and use the provided training and validation sets to compute errors. Report (1) the hyperparameter values that yield the best training error, and (2) the hyperparameter values that yield the best validation error. For each pair of values, train the model and visualize the decision boundaries using just the training set.

Note: on the real 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.

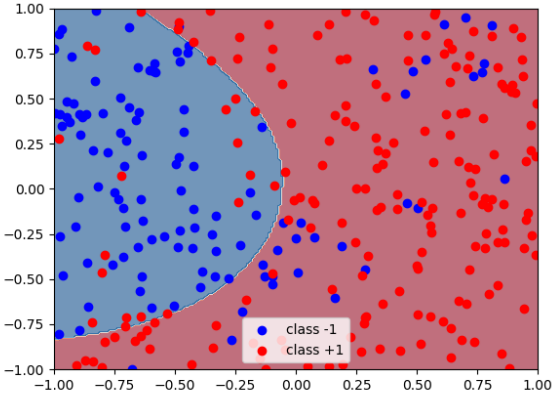
Answer: Again we'll accept the two implementations. For both implementations, the lowest training error is achieved with $\sigma = 10^{-2}$ and $\lambda = 10^{-4}$. Note to graders: some people reported having multiple combinations yielding the lowest training error. Accept any answer as long as the image looks good. Image below:



With the extra factor, the lowest validation error is achieved with $\sigma = 1$ and $\lambda = 0.1$. Figure below:



Without the extra factor, the lowest validation error is achieved with $\sigma = 0.1$ and $\lambda = 1$. The figure looks different; see below:



1.3 Reflection

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?

Answer: The results make sense: smaller σ leads to a more complex model, and smaller λ means less regularization; both of these should reduce training error. The validation error is minimized by larger values of these hyperparameters because the small values were overfitting. We can see in the plots that the first one is indeed a much more complex fit.

2 MAP Estimation

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

- The likelihood $p(y_i \mid x_i, w)$ 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.

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

   Answer:

$$f(w) = \|Xw - y\|_1 + \frac{1}{2\sigma^2}\|w\|^2.$$

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.

   Answer:

$$f(w) = \frac{1}{2}(Xw - y)^T\Sigma^{-1}(Xw - y) + \lambda\|w\|_1.$$

   The first term can also be written as $\frac{1}{2}\|\Sigma^{-\frac{1}{2}}(Xw - y)\|^2$.

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).

Answer:

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

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,

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

(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.)

Answer:

$$f(w) = \sum_{i=1}^{n}\left(-y_iw^Tx_i+\exp(w^Tx_i)\right).$$

If we wanted, we could write define $v_i = \exp(w^Tx_i)$ and write this in matrix notation as

$$f(w) = -y^TXw+1^Tv.$$

3 Principal Component Analysis

3.1 PCA by Hand

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?
2. What is the reconstruction loss (L2 norm squared) of the point (-3, 2.5)? (Show your work.)
3. What is the reconstruction loss (L2 norm squared) of the point (-3, 2)? (Show your work.)

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

Answer: The first variable has a mean of 0 so centering it does nothing. The mean of the second variable is 1 so the centered data looks like this:

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

1. We see that all the centered variables lie along the $x_2 = -0.5x_1$. One vector spanning this line would be $(2, -1)$ which has a norm of $\sqrt{5}$, and normalizing this vector gives $w_1 = (2/\sqrt{5}, -1/\sqrt{5})$ (the numbers could be expressed in other forms and could also have the opposite sign).

2. We first subtract the mean $(0, 1)$ to give $(-3, 1.5)$ and multiply by w_1,

$$z = -3 \cdot 2/\sqrt{5} + 1.5 \cdot (-1)/\sqrt{5} = -7.5/\sqrt{5}.$$

   To go back to the original space, we multiply this by w_1 and add back the means:

$$\hat{x} = -\frac{7.5}{\sqrt{5}}(2/\sqrt{5}, -1/\sqrt{5}) + (0, 1) = (-3, 1.5) + (0, 1) = (-3, 2.5),$$

   which is the same as the original point so the reconstruction loss is 0.

3. To get the low-dimensional representation, we first subtract the mean $(0, 1)$ to give $(-3, 1)$ and then multiply by w_1,

$$\begin{aligned} z &= -3 \cdot 2/\sqrt{5} + 1 \cdot (-1)/\sqrt{5} \\ &= -6/\sqrt{5} - 1/\sqrt{5} \\ &= -7/\sqrt{5}. \end{aligned}$$

   To go back to the original space, we multiply this by w_1 and add back the means:

$$\hat{x} = \frac{-7}{\sqrt{5}}(2/\sqrt{5}, -1/\sqrt{5}) + (0, 1) = (-2.8, 1.4) + (0, 1) = (-2.8, 2.4).$$

   so the reconstruction loss is

$$(-3 - (-2.8))^2 + (2 - 2.4)^2 = 0.2$$

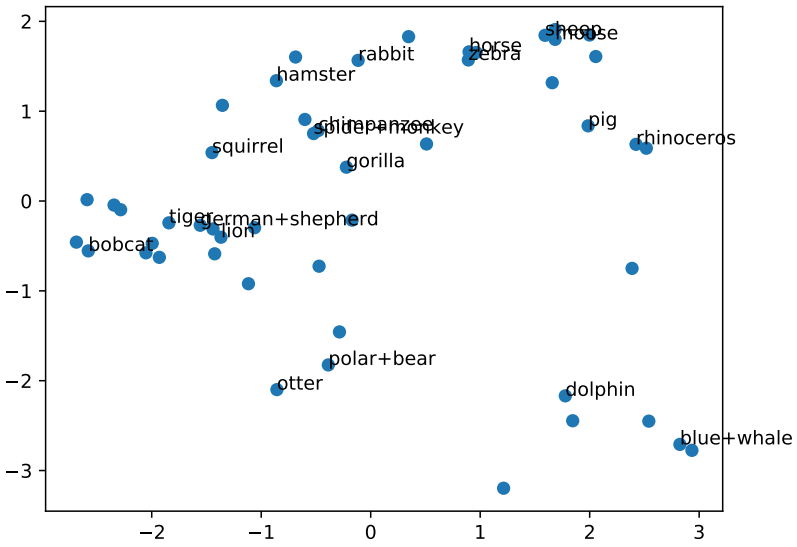
3.2 Data Visualization

If you run `python main.py -q 3.2`, the program will load a dataset containing 50 examples, each representing an animal. The 85 features are traits of these animals. The script standardizes these features and gives two unsatisfying visualizations of it. First it shows a plot of the matrix entries, which has too much information and thus gives little insight into the relationships between the animals. Next it shows a scatterplot based on two random features and displays the name of 20 randomly-chosen animals. Because of the binary features even a scatterplot matrix shows us almost nothing about the data.

In `compressors.py`, you will find a class named `PCA`, which implements the classic PCA method (orthogonal bases via SVD) for a given k, the number of principal components. Using this class, create a scatterplot that uses the latent features z_i from the PCA model with $k = 2$. Make a scatterplot of all examples using the first column of Z as the x-axis and the second column of Z as the y-axis, and use `plt.annotate()` to label a bunch of the points in the scatterplot. Do the following:

1. Hand in your modified demo and the scatterplot.

   Answer: My scatterplot looks like this:



2. Which trait of the animals has the largest influence (absolute value) on the first principal component?.

   Answer: With standardization, it's feature 22: "paws". Animals in the left side have paws, animals at the center have hooves, and animals in the right side have fins. Without standardization, it's "furry".

3. Which trait of the animals has the largest influence (absolute value) on the second principal component?

   Answer: Feature 54: "vegetation". Animals in the upper side are grazers, and animals in the lower side are predators.

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?

   Answer: It's important to center the data first before computing variance explained. Without standardization, I get 32.3% explained. With standardization, I get about 30%.

2. How many PCs are required to explain 50% of the variance in the data?

   Answer: $k \geq 5$.

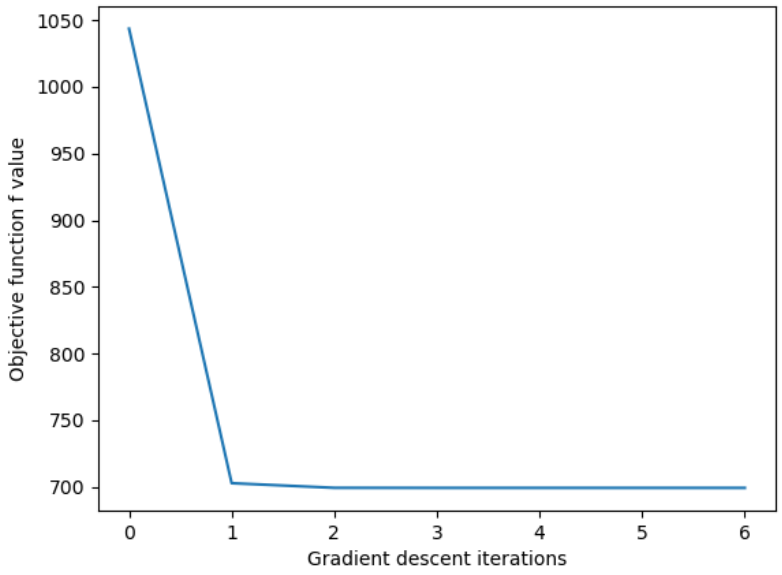
Note: you can compute the Frobenius norm of a matrix using the function *norm*. Also, note that the "variance explained" formula from class assumes that X is already centered.

4 Stochastic Gradient Descent

If you run `python main.py -q 4`, the program will do the following:

1. Load the dynamics learning dataset ($n = 10000, d = 5$)
2. Standardize the features
3. Perform gradient descent with line search to optimize an ordinary least squares linear regression model
4. Report the training error using `np.mean()`
5. Produce a learning curve obtained from training

The learning curve obtained from our `OptimizerGradientDescentLineSearch` looks like this:



This dataset was generated from a 2D bouncing ball simulation, where the ball is initialized with some random position and random velocity. The ball is released in a box and collides with the sides of the box, while being pulled down by the Earth's gravity. The features of X are the position and the velocity of the ball at some timestep and some irrelevant noise. The label y is the y-position of the ball at the next timestep. Your task is to train an ordinary least squares model on this data using stochastic gradient descent instead of the deterministic gradient descent.

4.1 Batch Size of SGD

In `optimizers.py`, you will find `OptimizerStochasticGradient`, a *wrapper* class that encapsulates another optimizer–let's call this a child optimizer. `OptimizerStochasticGradient` uses the child optimizer's `step()` method for each mini-batch to navigate the parameter space. The constructor for `OptimizerStochasticGradient` has two arguments: `batch_size` and `learning_rate_getter`. The argument `learning_rate_getter` is an object of class `LearningRateGetter` which returns the "current" value learning rate based on the number of batch-wise gradient descent iterations. Currently. `LearningRateGetterConstant` is the only class fully implemented.

Submit your code from `main.py` that instantiates a linear model optimized with `OptimizerStochasticGradient` taking a `OptimizerGradientDescent` (not line search!) as a child optimizer. Do the following:

1. Use ordinary least squares objective function (no regularization).
2. Using `LearningRateGetterConstant`, set the step size to $\alpha^t = 0.0003$.

3. Try the batch size values of `batch_size` $\in \{1, 10, 100\}$.

For each batch size value, use the provided training and validation sets to compute and report training and validation errors after 10 epochs of training. Compare these errors to the error obtained previously.

Answer: Depending on the random seed, the specific results will be different. For batch size of 1, I get training error=0.114 and validation error=0.115. For batch size of 10, I get training error=0.117 and validation error=0.118. For batch size of 100, I get training error=0.179 and validation error=0.180. It seems that stochastic gradient optimizer has not converged, but the small-batch results seem better than with the converged solution of the gradient descent optimizer.

4.2 Learning Rates of SGD

Implement the other unfinished `LearningRateGetter` classes, which should return the learning rate α^t based on the following specifications:

1. `LearningRateGetterConstant`: $\alpha^t = c$.
2. `LearningRateGetterInverse`: $\alpha^t = c/t$.
3. `LearningRateGetterInverseSquared`: $\alpha^t = c/t^2$.
4. `LearningRateGetterInverseSqrt`: $\alpha^t = c/\sqrt{t}$.

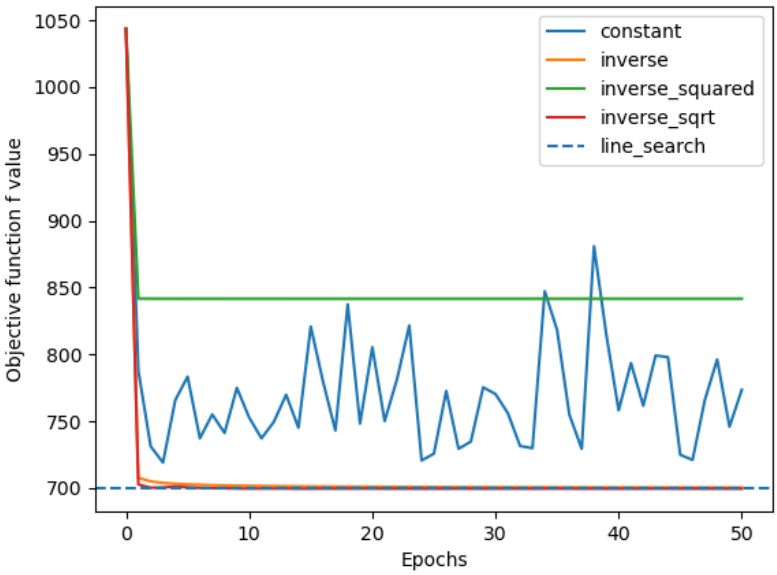
Submit your code for these three classes.

Answer: See code.

4.3 The Learning Curves (Again)

Using the four learning rates, produce a plot of learning curves visualizing the behaviour of the objective function f value on the y-axis, and the number of stochastic gradient descent epochs (50 or more epochs) on the x-axis. Use 10 as the batch size. For the value of c, use $c = 0.3$ for every learning rate function. Submit this plot and answer the following question. Which step size functions lead to the parameters converging towards a global minimum?

Answer: Only 2 and 4 (with 3 it will converge but not necessarily to a stationary point, with 1 it won't converge). 2 converges fairly quickly, but 4 decays slower. Here's the plot:



5 Very-Short Answer Questions

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?

   Answer: Likely faster if $n \ll d$.

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?

   Answer: Can find non-convex clusters.

3. In the language of loss functions and regularization, what is the difference between MLE and MAP?

   Answer: In MLE you don't have a regularizer.

4. What is the difference between a generative model and a discriminative model?

   Answer: Discriminative doesn't model probability of features X.

5. With PCA, is it possible for the loss to increase if k is increased? Briefly justify your answer.

   Answer: No, in the worse case you could always pick W to include the old W as a subspace, so the loss cannot be worse than it used to be with smaller k.

6. What does "label switching" mean in the context of PCA?

   Answer: You obtain the same model if you switch the order of the factors.

7. Why doesn't it make sense to do PCA with $k > d$?

   Answer: With $k = d$ you can already get an error of 0.

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

   Answer: A row of W.

9. What is an advantage and a disadvantage of using stochatic gradient over SVD when doing PCA?

   Answer: Stochastic gradient could be faster for huge datasets, SVD gives you the exact answer. SVD also gives orthonormal basis vectors.

10. 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?

    Answer: Global features let you predict for new users, local features let you make personalized predictions for individual users (if you have enough data for them).