**01 - Applying logistic regression and SVM**

**1. Welcome to the course!**

Welcome to the course on logistic regression and support vector machines with Python! In this first chapter, we'll cover the syntax for using these classifiers in scikit-learn. In Chapter 2, we'll go into a more conceptual study of loss functions. This will form the basis for going deeper into logistic regression and support vector machines (or SVMs) in Chapters 3 and 4.

**2. Assumed knowledge**

In this course we'll assume you've taken the prerequisite courses or have a similar level of knowledge. In this video we'll briefly review the standard syntax of the popular machine learning package scikit-learn, which was covered in the prerequisite course on supervised learning. We'll continue to use scikit-learn extensively in this course. To remind you, supervised learning refers to learning a relationship from examples of input-output pairs, usually called X and y.

**3. Fitting and predicting**

There are a few typical steps of supervised learning. First, let's load the newsgroups data from scikit-learn's repository of built-in datasets. We can inspect the shape of X and y and see that we have about 11,000 training examples, each with about 130,000 features. In this case the features are derived from the words appearing in each news article, and the y-values are the article topics, which is what we're trying to predict.

**4. Fitting and predicting (cont.)**

Next, we can import the k nearest neighbors classifier, or KNN for short. We instantiate the classifier, and store it in the variable knn. This is the step where we specify model hyperparameters, like the number of neighbors for KNN. Next, we can fit the model using the "fit" method. This is standard syntax across all of scikit-learn. Then, we can make predictions on any data set, including the original training set X. The variable y\_pred now contains one entry per row of X with the prediction from the trained classifier.

**5. Model evaluation**

Let's evaluate our KNN classifier. We can use the "score" function to compute the score on the training data, and it's tempting to be satisfied by almost 100% accuracy. But this number isn't particularly meaningful, since we want to know how the model generalizes to unseen data. This ability to generalize is often measured with a validation set. scikit-learn provides a convenient function to split up our data, train\_test\_split. X\_train and y\_train now contain the training set, and X\_test and y\_test now contain the test or validation set, which by default contains 25% of the examples. Before we compute the score on the test set, we need to make sure our model is training on the training set only, otherwise it would have access to the test set, defeating its purpose. So we refit on the training set, and compute the test score. We see that we have a much lower testing accuracy. Whether or not 66% accuracy is considered "good" depends on the situation, but the training error was definitely a poor representation of the model's ability to classify new data.

**6. Let's practice!**

Let's practice using KNN with scikit-learn.

**Got It!**

**1. Applying logistic regression and SVM**

In this video, we'll see how to run logistic regression and SVM with scikit-learn.

**2. Using LogisticRegression**

The LogisticRegression class in scikit-learn is used just like the other models you've seen in the prerequisite course. First, we import LogisticRegression from scikit learn. You'll notice we're importing from linear\_model, because logistic regression is a linear classifier. More on this later. Then, we instantiate an instance of the classifier. We fit the classifier on our training set. And then we can predict, compute the score, etc.

**3. LogisticRegression example**

Let's try this on an example data set, in this case the wine classification data set built into scikit-learn. We load the data set. Then, we create and fit a LogisticRegression object. We compute the training accuracy and see it's about 97%. scikit-learn's LogisticRegression can also output confidence scores rather than "hard" or definite predictions. Let's do this with the "predict\_proba" function and test it out on the first training example. Here the classifier is reporting over 99% confidence for the first class, and very low probabilities for the other two. As a reminder, the little e means "10 to the power of", so you should interpret that first probability as 9-point-9 times 10 to the power of -1, or point-99, or 99%. We'll discuss these probabilities more in Chapter 3.

**4. Using LinearSVC**

In scikit-learn, the basic SVM classifier is called LinearSVC for linear support vector classifier. The LinearSVC object works exactly the same way as LogisticRegression. Note that this data set has more than 2 classes. scikit-learn's Logistic Regression and SVM implementations handle this automatically. We'll talk about how this works in Chapter 3.

**5. Using SVC**

We can repeat these steps again for the "SVC" class, which fits a nonlinear SVM by default. As you can see, the classifier achieves 100% training accuracy. This could be the classifier overfitting, which is a risk we take when using more complex models like nonlinear SVMs. Later in this chapter, we'll discuss what it means for a classifier to be linear or not. By the way, so far we've used the default hyperparameters for LogisticRegression, LinearSVC, and SVC. To remind you, a hyperparameter is a choice about the model you make before fitting to the data, and often controls the complexity of the model. If the model is too simple, it may be unable to capture the patterns in the data, leading to low training accuracy; this is called underfitting. On the other hand, if the model is too complex it may learn the peculiarities of your particular training set, leading to lower test accuracy; this is called overfitting. This is a fundamental tradeoff in machine learning. In Chapters 3 and 4 we'll delve into these classifiers in more detail so that, by the end of the course, you'll understand what many of the hyperparameters represent, how they affect this fundamental tradeoff, and how to go about setting them.

**6. Let's practice!**

Now it's your turn to apply these classifiers.

**Got It!**

**1. Linear decision boundaries**

In this video, we'll discuss what it means for a classifier to be linear.

**2. Linear decision boundaries**

A decision boundary tells us what class our classifier will predict for any value of x. In this image, the classifier predicts the blue class in the blue shaded area, where feature 2 is small, and the red class in the red shaded area, where feature 2 is large. The dividing line between the two regions is called the decision boundary. This decision boundary is considered linear because it looks like a line. The line doesn't have to be horizontal; it could be in any orientation. This definition extends to more than 2 features as well. With 5 features, the space of possible x-values is 5-dimensional, which is hard for me to draw on a slide! In that case, the boundary would be a higher-dimensional "hyperplane" cutting the space into two halves. A nonlinear boundary is any other type of boundary. Sometimes this leads to non-contiguous regions of a certain prediction, like in the figure. In their basic forms, logistic regression and SVMs are linear classifiers, which means they learn linear decision boundaries. In Chapter 4 we'll discuss nonlinear versions that produce boundaries like the one on the right.

**3. Definitions**

Here's a list of some important vocabulary we'll be using in the course. Classification is supervised learning when the y-values are categories; this is in contrast with regression, where we're trying to predict a continuous value. We've defined decision boundaries and linear classifiers on the previous slide. A data set is called linearly separable if it can be perfectly explained by a linear classifier.

**4. Linearly separable data**

Here's an example of a data set that is linearly separable and one that is not. In the left figure, there's no single line that separates the red and blue examples. Note that for now we're just talking about binary, or 2-class, classification. Later in the course we'll discuss multi-class classification (with 3 or more categories). On the other hand, in the right-hand figure we could divide the two classes with a straight line, so it's called linearly separable.

**5. Let's practice!**

So if logistic regression, and linear SVMs are both linear classifiers, how are they different? That's what Chapter 2 is all about. Let's look at some more examples of linear and nonlinear decision boundaries.

**02 - Loss functions**

**1. Linear classifiers: prediction equations**

Welcome to Chapter 2. This chapter is much more conceptual than the other chapters, because we'll be laying the foundation for understanding logistic regression and SVMs. We'll start off by exploring some math behind linear classifiers in this video. By really digging into the details, you'll be better equipped to compare these classifiers to other models and interpret the results.

**2. Dot Products**

We'll start by defining a dot product. Let's create some numpy arrays x and y. To take the dot product between them, we need to multiply them element-wise. The result is 0 (from 0 times 3), 4 (from 1 times 4), and 10 (from 2 times 5). The sum of these numbers, also known as the dot product, is 14. A convenient notation for this in recent Python versions is the "at" symbol. x@y gives us the same result. In math notation, this is written "x dot y". You can think of a dot product as multiplication in higher dimensions, since x and y are arrays of values.

**3. Linear classifier prediction**

Using dot products, we can express how linear classifiers make predictions. First, we compute what we'll call the "raw model output", which is the dot product of the coefficients and the features, plus an intercept. We'll then take the sign of this quantity, in other words, we'll check if it's positive or negative. This is a key equation in the course. Crucially, this pattern is the same for both logistic regression and linear SVMs. In scikit-learn terms, we can say logistic regression and linear SVM have different fit functions but the same predict function. The differences in "fit" relate to loss functions, which are coming later in this chapter.

**4. How LogisticRegression makes predictions**

Let's see this equation in action with scikit-learn's breast cancer classification data set. We create a logistic regression object, fit it to the data, and look at the predictions on examples 10 and 20, which are 0 and 1.

**5. How LogisticRegression makes predictions (cont.)**

Let's now dig deeper. We can get the learned coefficients and intercept with lr.coef and lr.intercept. Let's compute the raw model output for example 10. It's negative: that's why we predict the negative class, called "0" in this data set. On the other hand, for example 20 the raw model output is positive: so we predict the other class, called "1" in this data set. In general, this is what the predict function does for any X: it computes the raw model output, checks if it's positive or negative, and then returns a result based on the names of the classes in your data set, in this case 0 and 1.

**6. The raw model output**

Let's look at our prediction equation visually. This figure shows an example in 2 dimensions, with the raw model output labeled at a few locations. As we move away from the boundary on one side, the output becomes more and more negative. On the other side, it becomes more and more positive. So the sign, positive or negative, tells you what side of the decision boundary you're on, and thus your prediction. Along the decision boundary itself, the raw model output is zero. Furthermore, the values of the coefficients and intercept determine the boundary.

**7. The raw model output**

For example, here I changed the intercept of the boundary. The raw model output at the same 3 points has changed because we changed the intercept. The boundary shifted down and left. To change the orientation of the boundary, we can change the coefficients.

**8. The raw model output**

Here we're looking at different coefficients. This changes the orientation of the decision boundary. In fact, the three points we were looking at are now all along the boundary.

**9. Let's practice!**

Let's continue exploring these effects.

**Got It!**

**1. What is a loss function?**

In this video, we'll kick off our discussion of loss functions. Many machine learning algorithms involve minimizing a loss, and by understanding this perspective you'll be equipped with the tools to see connections between models, quickly grasp new ones, and start tailoring them to your data science problem.

**2. Least squares: the squared loss**

We have actually seen loss functions before, in the prerequisite course on supervised learning. For example, least squares linear regression, such as scikit-learn's LinearRegression class, minimizes the sum of squares of the errors made on your training set. Here, error is defined as the difference between the true target value and the predicted target value. You can think of minimizing the loss as jiggling around the coefficients, or parameters of the model until this error term, or loss function, is as small as possible. In other words, the loss function is a penalty score that tells us how well (or, to be precise, how poorly) the model is doing on the training data. We can think of the "fit" function as running code that minimizes the loss. Note that the score function provided by scikit-learn isn't necessarily the same thing as the loss function. The loss is used to fit the model on the data, and the score is used to see how well we're doing. It's intuitive that these would be the same, and they often are, but you should be aware that this isn't guaranteed.

**3. Classification errors: the 0-1 loss**

The squared error from LinearRegression is not appropriate for classification problems, because our y-values are categories, not numbers. For classification, a natural quantity to think about is the number of errors we've made. Since we'd like to make this as small as possible, the number of errors might be a good loss function. We'll refer to this loss function as the 0-1 loss, because it's defined to be either 0 (if your prediction is correct) or 1 (if your prediction is wrong). By summing this function over all training examples, we get the number of mistakes we've made on the training set, since we add 1 to the total for each mistake. While the 0-1 loss is important for our conceptual journey, it turns out to be very hard to minimize it directly in practice, which is why logistic regression and SVMs don't use it. The reasons for this are beyond the scope of the course.

**4. Minimizing a loss**

In the exercises you'll try minimizing a loss function using a Python package called scipy-dot-optimize-dot-minimize, which can minimize all sorts of functions. Let's try it out. Here, I'll minimize the function y=x^2, which is computed using numpy.square. The second argument is our initial guess. Let's try zero. Finally, I have "dot x" at the end to grab the input value that makes the function as small as possible. We got zero as a result because this function is minimized when x=0. But that's not too interesting, since our initial guess was the correct answer! It's correct because something squared can only be zero or more, thus, its smallest possible value is attained when x=0. Let's try another initial guess to see if it's actually doing something. What we see is a very small number, near 10 to the power of -8. This is normal for numerical optimization: we don't expect exactly the right answer, but something very close. In the exercises, you'll minimize the squared error from linear regression. The inputs will be the model coefficients. So, you can think of the code as answering the question, "what values of the model coefficients make my squared error as small as possible?" That's exactly what linear regression is doing.

**5. Let's practice!**

Time for a couple exercises on loss functions.

**03 - Logistic regression**

**1. Logistic regression and regularization**

Welcome to Chapter 3! In this chapter we'll use the practical skills from Chapter 1 and the concepts from Chapter 2 to dig deeper into logistic regression.

**2. Regularized logistic regression**

The prerequisite course, "supervised learning with scikit-learn", mentions that regularization combats overfitting by making the model coefficients smaller. The figure shows the learned coefficients of a logistic regression model with default regularization. In scikit-learn, the hyperparameter "C" is the inverse of the regularization strength. In other words, larger C means less regularization and smaller C means more regularization. Let's test this out.

**3. Regularized logistic regression**

The orange curve shows what happens if we use a smaller value of C, which means more regularization for our logistic regression model. As expected, regularization makes the coefficients smaller.

**4. How does regularization affect training accuracy?**

Let's see how regularization influences training and test accuracy. With the movie review data set already loaded and split into train and test sets, we instantiate two logistic regression models, one with weak regularization and one with strong regularization. We then fit both models. Next, we compute training accuracy. The model with weak regularization gets a higher training accuracy. Now that we've studied loss functions, we can see why regularization makes the training accuracy go down: Regularization is an extra term that we add to the original loss function, which penalizes large values of the coefficients. Intuitively, without regularization, we are maximizing the training accuracy, so we do well on that metric. When we add regularization, we're modifying the loss function to penalize large coefficients, which distracts from the goal of optimizing accuracy. The larger the regularization penalty (or the smaller we set C), the more we deviate from our goal of maximizing training accuracy. Hence, training accuracy goes down.

**5. How does regularization affect test accuracy?**

Let's look at the test accuracy this time. As we can see, regularization improved it. We discussed why regularization reduces training accuracy, but why does it improve test accuracy? Imagine you did not have access to a particular feature; that's like setting the corresponding coefficient to zero. Regularizing, and thus making your coefficient smaller, is like a compromise between not using the feature at all (setting the coefficient to zero) and fully using it (the un-regularized coefficient value). If using a feature too heavily was causing overfitting, then regularization causes you to "fit less" and thus overfit less.

**6. L1 vs. L2 regularization**

For linear regression we use the terms Ridge and Lasso for two different types of regularization. The general names for these concepts, outside linear regression, are L1 regularization and L2 regularization. Everything you learned about ridge (or L2) and lasso (or L1) in the past applies to logistic regression as well. For example, both help reduce overfitting, and L1 also performs feature selection. As an example, let's train two logistic regression models, with L1 and L2 regularization, on the breast cancer dataset after scaling features, which is usually good practice, especially when using regularization. We can plot the coefficients for both models, adding a grid so that we can see where zero is.

**7. L2 vs. L1 regularization**

Here are the plots. As you can see, L1 regularization set many of the coefficients to zero, thus ignoring those features; in other words, it performed feature selection for us. On the other hand, L2 regularization just shrinks the coefficients to be smaller. This is analogous to what happens with Lasso and Ridge regression.

**8. Let's practice!**

Now it's your turn to explore regularization for logistic regression.

**Got It!**

**1. Logistic regression and probabilities**

So far we've been using logistic regression to make hard predictions, meaning we predict either one class or the other. In this video we'll discuss how to interpret the raw model output of the classifier as a probability.

**2. Logistic regression probabilities**

We've seen this type of decision boundary several times in the course. The fill color shows what class we would predict for every point in the space. In Chapter 1 we saw that the scikit-learn logistic regression object can output probabilities with the "predict\_proba" function. Let's make the same type of figure but this time showing the probabilities.

**3. Logistic regression probabilities**

In this figure, the new interpretation of the colors is the predicted probability of the red class. The black line is the old decision boundary, which we can refer to if we need to make definite, or hard, decisions. We can see that this line is where the probabilities cross point-5. In other words, if we're more than 50% sure it's red, we predict red, and if we're less than 50% sure it's red, we predict blue. We can also see that we get more and more confident as we move away from the decision boundary, which sounds reasonable. In this figure, regularization is effectively disabled because C is very large.

**4. Logistic regression probabilities**

The figure on the right shows what happens when we turn on regularization. First, we see that the coefficients are smaller, as expected. The effect of regularization is that the probabilities are closer to point-5; we don't get to the very dark red or very dark blue on the right-hand figure. In other words, smaller coefficients mean less confident predictions. This fits with our story: regularization is supposed to combat overfitting, and there's a connection between overconfidence and overfitting. By the way, these figures also answer a question you may have had from the previous chapter. With 2 features, we had 2 coefficients even though you only really need one number to represent the slope of a line. We now have a reason for this: the ratio of the coefficients gives us the slope of the line, and the magnitude of the coefficients gives us our confidence level. Finally, as you can see, regularization not only affects the confidence, but also the orientation of the boundary.

**5. How are these probabilities computed?**

So how are these probabilities computed? Like the definite class predictions, they come from the raw model output. The raw model output can be any number, but probabilities are numbers between 0 and 1. So we need a way to "squash" the raw model output to be between 0 and 1. The sigmoid function takes care of that for us. Here's what it looks like. Take a look at the curve: when the raw model output is zero, the probability is point-5 - this means we're right on the boundary. When the raw model output is positive, we would have predicted the positive class, and indeed the probability of the positive class approaches 1. When the raw model output is negative, we would have predicted the negative class, and indeed the probability of the positive class approaches 0, which is another way of saying that we're very confident it's the negative class. Since the raw model output grows as we move away from the boundary, we're more confident in our predictions far away from the boundary.

**6. Let's practice!**

Time to explore these predicted probabilities.

**Got It!**

**1. Multi-class logistic regression**

Multi-class classification means having more than 2 classes. While we've used scikit-learn to perform multi-class classification, all of our conceptual discussions have been in the binary, or 2-class, case. In this video we'll discuss how multi-class classification works for linear classifiers.

**2. Combining binary classifiers with one-vs-rest**

We'll cover two popular approaches to multi-class classification. The first is to train a series of binary classifiers for each class. For example, I've loaded the wine dataset and instantiated 3 logistic regression classifiers. I'll now fit these classifiers on 3 different data sets. The code y==0 returns an array the same size as y that's True when y is 0 and False otherwise, so the classifier learns to predict these true/false values. In other words, it's a binary classifier learning to discriminate between class 0 or not 0. The next one learns y=1 vs. not 1, etc. This is called the one-vs-rest strategy. In order to make predictions using one-vs-rest, we take the class whose classifier gives the largest raw model output - or decision\_function, in scikit-learn terminology. In this case, the largest raw model output comes from classifier 0. This means it's more confident that the class is 0 than any of the other classes, so we predict class 0. One-vs-rest is the default behavior of scikit-learn's LogisticRegression. We can just let scikit-learn do the work by fitting a logistic regression model on the original multi-class data set. We get the same prediction, of 0, as expected.

**3. One-vs-rest vs. multinomial/softmax**

Another way to achieve multi-class classification with logistic regression is to modify the loss function so that it directly tries to optimize accuracy on the multi-class problem. You may encounter various words related to this, like multinomial logistic regression, softmax, or cross-entropy loss. The slide shows a comparison of the two approaches. In one case you fit separately for each class, whereas in the other you just do it once. The same goes for prediction. An appealing property of the binary approach is that you can reuse your binary classifier implementation rather than needing a new one. On the other hand, you might sometimes get better accuracy with the multinomial classifier since its loss is more directly aligned with accuracy. In the field of neural networks, the multinomial approach is standard. Finally, while both approaches can work for SVMs, one-vs-rest and related strategies tend to be more popular. By the way, both of these methods can output probabilities just like a binary classifier.

**4. Model coefficients for multi-class**

We've talked a lot about the coefficients, so it's natural to ask, what do the coefficients look like for multi-class classification? Continuing with the wine dataset, let's fit a one-vs-rest model and look at the coefficients. In the binary case we had one coefficient per feature and one intercept. For 3 classes we now have 3 entire binary classifiers, so we end up with one coefficient per feature per class, and one intercept per class. Hence, the coefficients of this model are stored in a 3-by-13 array. We can instantiate the multinomial version by setting the mutli\_class argument. In scikit-learn, this also requires changing to a non-default solver like "lbfgs". The solver hyperparameter specifies the algorithm used to minimize the loss; the default algorithm is for the binary problem, so it can be used for one-vs-rest but not multinomial. As we can see, The multinomial classifier has the same number of coefficients and intercepts as one-vs-rest. Although these two approaches work differently, they learn the same number of parameters and, roughly speaking, the parameters have the same interpretations.

**5. Let's practice!**

Your turn to explore these two multi-class approaches.

**Support Vector Machines**

**1. Support Vectors**

Welcome to the final chapter of the course, where we'll discuss SVMs in more detail. In this first video we'll discuss what support vectors are and why they matter.

**2. What is an SVM?**

In the last chapter we talked about logistic regression, which is a linear classifier learned with the logistic loss function. Linear SVMs are also linear classifiers, but they use the hinge loss instead. The standard definition of an SVM also includes L2 regularization. Remember these diagrams from Chapter 2? The logistic and hinge losses look fairly similar. A key difference is in the "flat" part of the hinge loss, which occurs when the raw model output is greater than 1, meaning you predicted an example correctly beyond some margin of error. If a training example falls in this "zero loss" region, it doesn't contribute to the fit; if I removed that example, nothing would change. This is a key property of SVMs.

**3. What are support vectors?**

Support vectors are defined as examples that are NOT in the flat part of the loss diagram. In the figure, support vectors are shown with yellow circles around them. Another way of defining support vectors is that they include incorrectly classified examples, as well as correctly classified examples that are close to the boundary. If you're wondering how close is considered close enough, this is controlled by the regularization strength. Support vectors are the examples that matter to your fit. If an example is not a support vector, removing it has no effect on the model, because its loss was already zero. Even though we use the name "support vectors", it's really the non-support-vectors that are remarkable. Comparing with logistic regression, there is no flat part of the loss there, and therefore all data points matter to the fit. Critical to the popularity of SVMs is that kernel SVMs, coming later in this chapter, are surprisingly fast to fit and predict. Part of the speed comes from clever algorithms whose running time only scales with the number of support vectors, rather than the total number of training examples.

**4. Max-margin viewpoint**

Although it's not the perspective we've taken in this course, you may encounter the idea that SVMs "maximize the margin". I want to briefly mention this viewpoint for completeness. The diagram shows an SVM fit on a linearly separable dataset. As you can see, the learned boundary falls just half way between the two classes. This is an appealing property: in the absence of other information, this boundary makes more sense than a boundary that is much closer to one class than the other.

**5. Max-margin viewpoint**

The yellow lines show the distances from the support vectors to the boundary. The length of the yellow lines, which is the same for all 3 cases, is called the margin. If the regularization strength is not too large, SVMs maximize the margin of linearly separable datasets. Unfortunately, most datasets are not linearly separable; in other words, we don't typically expect a training accuracy of 100%. While these max margin ideas can be extended to non-separable data, we won't pursue that avenue here. You can think of this as another view on what we've already defined SVMs to be, which is the hinge loss with L2 regularization. As it turns out, they are mathematically equivalent.

**6. Let's practice!**

Time play with support vectors.

**1. Kernel SVMs**

In this video we'll see how to fit nonlinear boundaries using linear classifiers.

**2. Transforming your features**

Consider this 2D toy dataset. The two classes are not linearly separable; in other words, there's no linear boundary that perfectly classifies all the points. If you try fitting a linear SVM on these points, you might get back something that predicts blue everywhere.

**3. Transforming your features**

However, notice that the red points are all close to the point (0,0) in the coordinate system. Let's create two new features, one of which is feature 1 squared and the other of which is feature 2 squared. That means values near zero will become small values, and values far from zero, both positive and negative, will become large. What happens now if we plot the points?

**4. Transforming your features**

Well, now they are linearly separable in this transformed universe, because all the red points are near the lower left and all the blue points are above and to the right. We can fit a linear SVM using these new features and

**5. Transforming your features**

the result is a perfect classification. But then we might ask ourselves, what does this linear boundary look like back in the original space. In other words, if we took these axes and un-squared them, what would happen to the shape of the boundary?

**6. Transforming your features**

In this case, we get an ellipse. So, what's the take-home message here? It's that fitting a linear model in a transformed space corresponds to fitting a nonlinear model in the original space. Nice! In general, the transformation isn't always going to be squaring and the boundary isn't always going to be an ellipse. In fact, the new space often has a different number of dimensions from the original space! But this is the basic idea. Kernels and kernel SVMs implement feature transformations in a computationally efficient way.

**7. Kernel SVMs**

Let's look at some code. We'll need to use scikit-learn's SVC class, rather than LinearSVC, to allow for different kernels. The default behavior is what's called an RBF or Radial Basis Function kernel. Although it's not computed this way, you can think of this as an extremely complicated transformation of the features, followed by fitting a linear boundary in that new space, just like we saw for the simpler squaring transformation. While many nonlinear kernels exist, in this course we'll focus on the RBF. With kernel SVMs, we can call fit and predict in all the usual ways. Let's look at a decision boundary. This is definitely not linear! And, as a result, we've gotten a higher training accuracy than we could have with a linear boundary. We can control the shape of the boundary using the hyperparameters. As usual we have the C hyperparameter that controls regularization. The RBF kernel also introduces a new hyperparameter, gamma, which controls the smoothness of the boundary. By decreasing gamma, we can make the boundaries smoother.

**8. Kernel SVMs**

The second image shows the same data set with gamma set to point-01. The boundary looks smoother.

**9. Kernel SVMs**

The third image shows gamma=2. Now we've reached 100% training accuracy by creating a little "island" of blue around each blue training example. In fact, with the right hyperparameters, RBF SVMs are capable of perfectly separating almost any data set. So, why not always use the largest value of gamma and get the highest possible training accuracy? You guessed it: overfitting. In the exercises you'll explore how the kernel hyperparameters affect the tradeoff between training and test accuracy.

**10. Let's practice!**

Time to experiment with kernel SVMs

**1. Comparing logistic regression and SVM**

In this video we'll compare our two linear classifiers, logistic regression and SVMs.

**2. Pros and cons**

Let's summarize the points we've covered throughout the course. Both logistic regression and SVM are linear classifiers. Both can be used with kernels, but this is more common with SVMs because the predictions are much faster when the number of support vectors is small. While both can be coerced to output probabilities, this is much more natural with logistic regression. Both can be extended to multi-class with a one-vs-rest scheme or by directly modifying the loss. In logistic regression, like most methods, all data points affect the fit. SVMs have the special property that only a subset of the examples are "support vectors" and the rest can be removed without affecting the fit. While the term "logistic regression" doesn't refer to a particular type of regularization, the term "SVM" typically refers to a linear classifier that uses the hinge loss and L2 regularization.

**3. Use in scikit-learn**

Let's compare the use of our two methods in scikit-learn. Logistic regression is imported via sklearn.linear\_model.LogisticRegression. And what are its key hyperparameters? The first is C, which controls the amount of regularization: smaller C means more regularization and vice versa. The next is the type of regularization: scikit-learn supports L2 and L1. We also discussed methods for extending a binary classifier to multi-class. This hyperparameter will probably affect your results less than the previous two. There are a bunch more hyperparameters that scikit-learn exposes, but these are the ones I consider the most fundamental. Don't be afraid to read about the rest! As for SVMs, they can be instantiated from sklearn.svm using either LinearSVC for a linear SVM or SVC for a kernel SVM. (You can actually also fit linear SVMs via the SVC class by setting the kernel to linear, but you may find LinearSVC to be faster.)

**4. Use in scikit-learn (cont.)**

The key hyperparameters of the SVC class are C, just like with logistic regression. the type of kernel - we only talked about linear and RBF, but scikit learn supports a couple others. and gamma, which only applies to the RBF kernel and controls the smoothness. Smaller values of gamma lead to smoother, or simpler, decision boundaries, and larger values of gamma lead to more complex boundaries. As with LogisticRegression, there are certainly more hyperparameters and I encourage you to check them out.

**5. SGDClassifier**

Finally, I want to direct your attention to scikit-learn's SGDClassifier. SGD stands for stochastic gradient descent. Although we didn't cover SGD in this course, it's worth knowing about SGDClassifier, since it can handle very large datasets much better than the other methods we've discussed. We've been talking about how logistic regression and SVM are just two types of linear classifiers, and SGDClassifier really brings this point home. In fact, to switch between logistic regression and a linear SVM, one only has to set the loss hyperparameter of the SGDClassifier. It's just like we said: the model is the same, and only the loss changes. SGDClassifier works pretty much like the other scikit-learn methods we've seen. One "gotcha" with SGDClassifier is that the regularization hyperparameter is called alpha instead of C, and bigger alpha means more regularization. Basically, alpha is the inverse of C.

**6. Let's practice!**

Now it's your turn.

**How does this course fit into data science?**

You now have practice applying logistic regression and support vector machines to classification problems. How does this fit into a bigger picture? The way I see it, data science is the process of answering questions and making decisions based on data. The data science process usually combines several of the following pieces: data collection, data preparation, database design, visualization, communication, software engineering, machine learning, and more. In this course we focused on the machine learning portion. Machine learning has several branches like supervised learning, unsupervised learning, and reinforcement learning. We've been focusing on supervised learning, which means trying to predict a target value from features given a labeled data set. Within supervised learning, we've focussed on classification, which means the thing we're trying to predict is categorical rather than a continuous in nature. Finally, we covered logistic regression and SVMs, the two most popular linear classifiers. So, while we've done a lot, there's so much more out there!