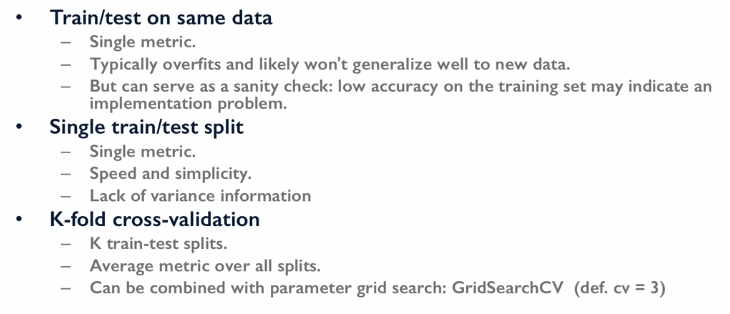
**Optimizing Classifiers for Different Evaluation Metrics:**

The general workflow of developing a machine learning algorithm is as seen below, however, remember that this process is iterative and often after evaluating a model you will have to go back and carry out some feature engineering to improve certain outcomes of your model.

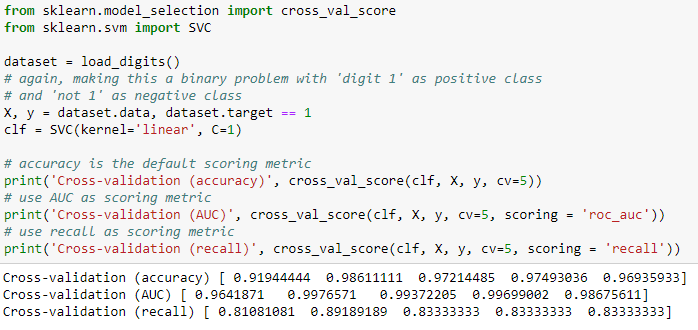
It’s also important to note that the best practise of machine learning is to split your data into **3 parts**, not just 2. Where the first 2 will be used for cross validation and tuning and the final one will be used for only evaluation and **no further tuning is allowed!**



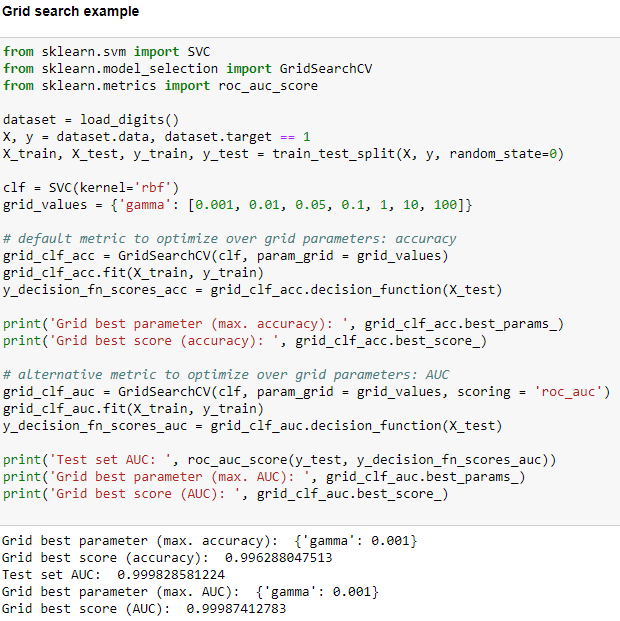
If we just use a simple training and testing dataset it would typically result in overfitting and wouldn’t generalize well to new data points. However, this is often a good initial step to ensure that code and feature engineering is working properly. This single use of the training and testing data gives a fast and easy single evaluation of the model. But this doesn’t give a realistic outlook on the model, it also doesn’t give us a good understanding of the variance the model might produce. The third step we could use is k-fold cross-validation which gives a more reliable prediction of how our model will perform on unseen data.

We can go further and use grid search with cross-validation to find optimal parameters for a model with respect to the evaluation metric. The default evaluation metric used for a cross-val score or GridSearchCV is accuracy. So how do you apply the new metrics you've learned about here like AUC in model selection? Scikit-learn makes this very easy. You simply add a scoring parameter that's set to the string with the name of the evaluation metric you want to use.

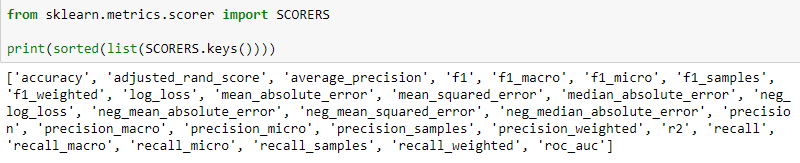
The below code shows how we can use cross-validation and a desired scoring parameter to return the desired score:



In the below example we’re using a **grid search** and **cross validation** to return the **optimum hyperparameters**:



To see all the evaluation metrics supported by model selection we can use the code below:

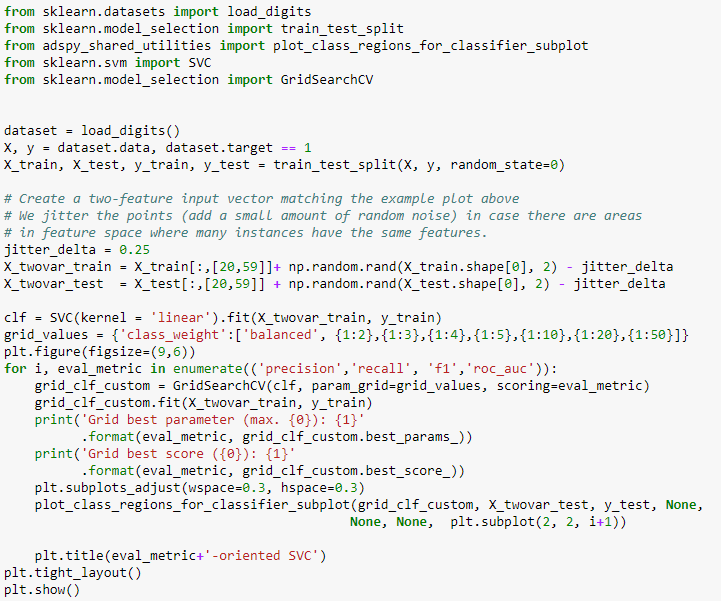


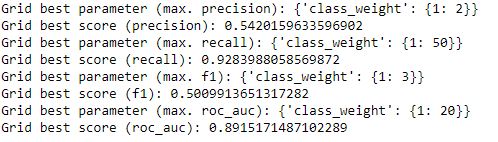
**Example: Optimizing a Classifier Using Different Evaluation Metrics:**

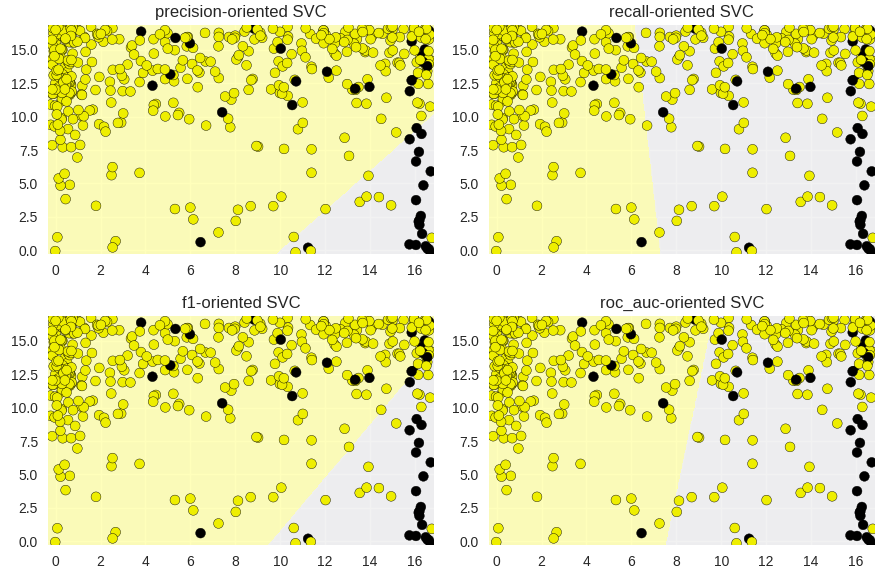
Models will have to be optimized for different metrics. For example, a model that predicts if someone has Covid will have to have a very high recall as any false negatives would cause the virus to spread faster. Or perhaps we’re making a video recommendation system that needs to be very precise to ensure customer satisfaction. Another example would be predicting handwritten digits, for this type of problem over all accuracy is the most important parameter.

In the below example the **data has class imbalances** (the positive class is much smaller), so we have used class weights to try and reduce the bias in the data.

We apply grid search here to explore different values of the optional **class weight parameter** that controls how much weight is given to each of the two classes during training. As it turns out**, optimizing for different evaluation metrics results in different optimal values of the class weight parameter**. As the class weight parameter increases, more emphasis will be given to correctly classifying the positive class instances.



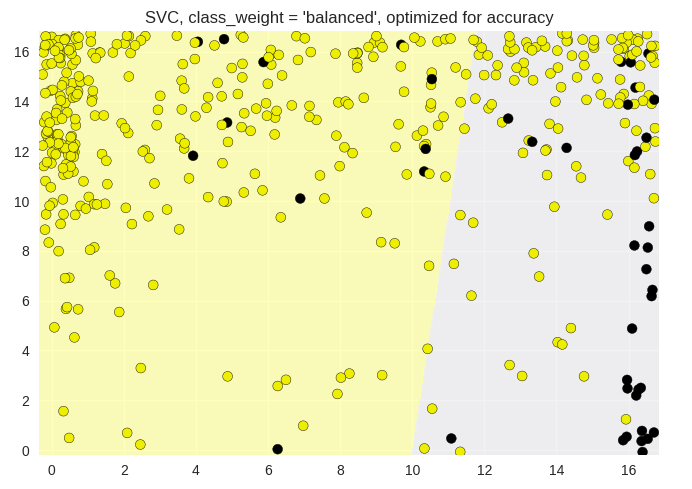




The **precision-oriented** **classifier** we see here with **class weight of two**, tries hard to reduce **false positives** while increasing **true positives**. So, it focuses on the cluster of positive class points in the lower right corner where there are relatively few negative class points. Here, precision is over 50 percent. In contrast, the **recall-oriented** **classifier** with **class weight of 50**, tries hard to reduce the number of **false negatives** while increasing **true positives**. That is, it tries to find most of the positive class points as part of its positive class predictions. We can also see that the decision boundary for the **F1-oriented** **classifier** has an optimal class weight of two, which is between the optimal class weight values for the precision and recall-oriented classifiers. Visually we can see that the F1-oriented classifier also has a kind of intermediate positioning between the precision and recall-oriented, decision boundaries. This makes sense given that **F1 is the harmonic mean of precision and recall**. The AUC-oriented classifier with optimal class weight to 5 has a similar decision boundary to the F1-oriented classifier but shifted slightly in favour of higher recall.

If we only care about **accuracy**, we can set the **class weights to be balanced**.

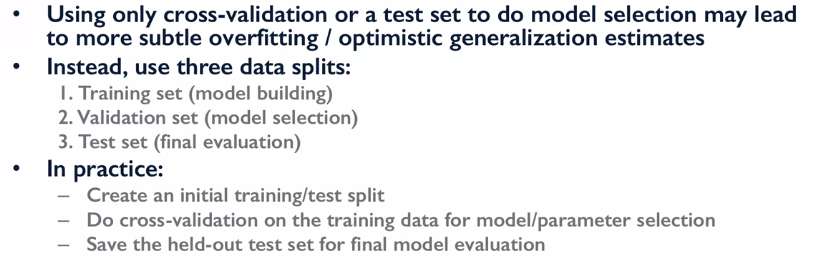


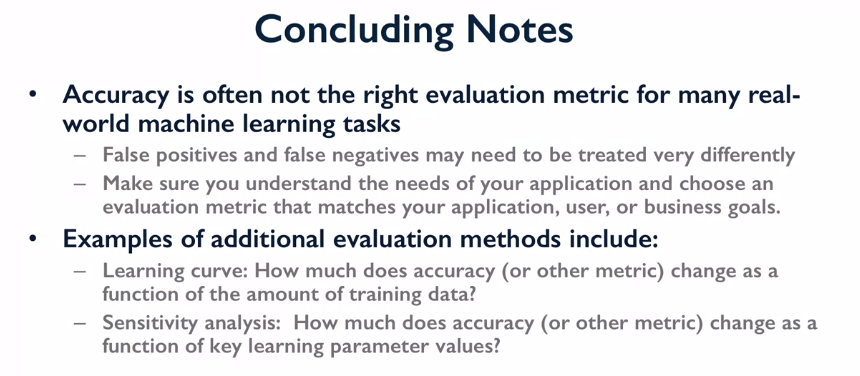




We can see the **precision recall trade-off** very clearly for this classification scenario in the precision recall curve for the default support vector classifier with linear kernel optimized for accuracy on the same dataset and using the balanced option for the class weight parameter. This makes sense as there are far more negative classes and there for the chance of false negatives becomes more likely so improving recall is heavily linked to overall accuracy.

**Why we need to split the data into 3 set:**





**Quiz Question:**

