Facies classification using machine learning

Brendon Hall[1]

There has been much excitement recently about *big data* and the dire need for *data scientists* who possess the ability to extract meaning from it. Geoscientists, meanwhile, have been doing science with voluminous data for years, without needing to brag about how big it is. But now that large, complex data sets are widely available, there has been a proliferation of tools and techniques for analyzing them. Many free and open-source packages now exist that provide powerful additions to the geoscientist's toolbox, much of which used to be only available in proprietary (and expensive) software platforms.

One of the best examples is `scikit-learn` (http://scikit-learn.org/), a collection of tools for *machine learning* in Python. What is machine learning? You can think of it as a set of data-analysis methods that includes classification, clustering, and regression. These algorithms can be used to discover features and trends within the data without being explicitly programmed, in essence *learning* from the data itself.

In this tutorial, we will demonstrate how to use a classification algorithm known as a support vector machine to identify lithofacies based on well-log measurements. A support vector machine (or SVM) is a type of supervised-learning algorithm, which needs to be supplied with training data to learn the relationships between the measurements (or features) and the classes to be assigned. In our case, the features will be well-log data from nine gas wells. These wells have already had lithofacies classes assigned based on core descriptions. Once we have trained a classifier, we will use it to assign facies to wells that have not been described.

Exploring the data set

The data set we will use comes from a University of Kansas class exercise on the Hugoton and Panoma gas fields. For more on the origin of the data, see Dubois et al. (2007) and the Jupyter notebook that accompanies this tutorial at http://github.com/seg.

The data set consists of seven features (five wireline log measurements and two indicator variables) and a facies label at half-foot depth intervals. In machine learning terminology, the set of measurements at each depth interval comprises a *feature vector*, each of which is associated with a *class* (the facies type). We will use the `pandas` library to load the data into a dataframe, which provides a convenient data structure to work with well-log data.

```
>>> import pandas as pd
>>> data = pd.read_csv('training_data.csv')
```

We can use `data.describe()` to provide a quick overview of the statistical distribution of the training data (Table 1).

We can see from the `count` row in Table 1 that we have a total of 3232 feature vectors in the data set. The feature vectors consist of the following variables:

1) Gamma ray (GR)
2) Resistivity (ILD_log10)
3) Photoelectric effect (PE)
4) Neutron-density porosity difference (DeltaPHI)
5) Average neutron-density porosity (PHIND)
6) Nonmarine/marine indicator (NM_M)
7) Relative position (RELPOS)

There are nine facies classes (numbered 1–9) identified in the data set. Table 2 contains the descriptions associated with these classes. Note that not all of these facies are completely discrete; some gradually blend in to one another. Misclassification of these neighboring facies can be expected to occur. The `Adjacent Facies` column in Table 2 lists these related classes.

To evaluate the accuracy of the classifier, we will remove one well from the training set so that we can compare the predicted and actual facies labels.

Table 1. Statistical distribution of the training data set.

	Facies	Depth	GR	ILD_ log10	DeltaPHI	PHIND	PE	NM_M	RELPOS
count	3232	3232	3232	3232	3232	3232	3232	3232	3232
mean	4.42	2875.82	66.14	0.64	3.55	13.48	3.73	1.50	0.52
std	2.50	131.00	30.85	0.24	5.23	7.70	0.89	0.50	0.29
min	1	2573.50	13.25	-0.03	-21.83	0.55	.020	1	0.01
25%	2	2791.00	46.92	0.49	1.16	8.35	3.10	1	0.27
50%	4	2932.50	65.72	0.62	3.5	12.15	3.55	2	0.53
75%	6	2980.00	79.63	0.81	6.43	16.45	4.30	2	0.77
max	9	3122.50	361.15	1.48	18.60	84.40	8.09	2	1.00

[1]Enthought.

http://dx.doi.org/10.1190/tle35100906.1.

Table 2. Facies labels with their descriptions.

Facies	Description	Label	Adjacent facies
1	Nonmarine sandstone	SS	2
2	Nonmarine coarse siltstone	CSiS	1,3
3	Nonmarine fine siltstone	FSiS	2
4	Marine siltstone and shale	SiSh	5
5	Mudstone	MS	4,6
6	Wackestone	WS	5,7,8
7	Dolomite	D	6,8
8	Packstone-grainstone	PS	6,7,9
9	Phylloid-algal bafflestone	BS	7,8

```
>>> test_well = data[data['Well Name'] == 'SHANKLE']
>>> data = data[data['Well Name'] != 'SHANKLE']
```

Let's extract the feature vectors and the associated facies labels from the training data set:

```
>>> features = ['GR', 'ILD_log10', 'DeltaPHI',
        'PHIND','PE','NM_M', 'RELPOS']
>>> feature_vectors = data[features]
>>> facies_labels = data['Facies']
```

Crossplots are a familiar tool to visualize how two properties vary with rock type. This data set contains five log measurements, and we can employ the very useful `seaborn` library (Waskom et al., 2016) to create a matrix of crossplots to visualize the variation between the log measurements in the data set.

```
>>> import seaborn as sns
>>> sns.pairplot(feature_vectors[['GR', 'ILD_log10',
        'DeltaPHI', 'PHIND','PE']])
```

Each pane in Figure 1 (next page) shows the relationship between two of the variables on the *x* and *y* axis, with a stacked bar plot showing the distribution of each point along the diagonal. Each point is colored according to its facies (see the Jupyter notebook associated with this tutorial for more details on how to generate colors for this plot). It is not clear from these crossplots what relationships exist between the measurements and facies labels. This is where machine learning will prove useful.

Conditioning the data set

Many machine-learning algorithms assume the feature data are normally distributed (i.e., Gaussian with zero mean and unit variance). Table 1 shows us that this is not the case with our training data. We will condition, or *standardize*, the training data so that it has this property. The same factors used to standardize the training set must be applied to any subsequent data set that will be classified. `Scikit-learn` includes a handy `StandardScalar` class that can be applied to the training set and later used to standardize any input data.

```
>>> from sklearn.preprocessing import StandardScaler
>>> scaler = StandardScaler().fit(feature_vectors)
>>> scaled_features = scaler.transform(feature_vectors)
```

A standard practice when training supervised-learning algorithms is to separate some data from the training set to evaluate the accuracy of the classifier. We have already removed data from a single well for this purpose. It is also useful to have a *cross-validation* data set we can use to tune the parameters of the model. `Scikit-learn` includes a handy function to randomly split the training data into subsets. Let's use 5% of the data for the cross-validation set.

```
>>> from sklearn.cross_validation import train_test_split
>>> X_train, X_cv, y_train, y_cv = \
        train_test_split(scaled_features, facies_labels,
            test_size=0.05, random_state=42)
```

Training the classifier

Now we can use the conditioned data set to train a support vector machine to classify facies. The concept of a support vector

Can you do better?

I hope you enjoyed this month's tutorial. It picks up on a recent wave of interest in artificial-intelligence approaches to prediction. I love that Brendon shows how approachable the techniques are — the core part of the process only amounts to a few dozen lines of fairly readable Python code. All the tools are free and open source; it's just a matter of playing with them and learning a bit about data science.

In the blind test, Brendon's model achieves an accuracy of 43% with exact facies. We think the readers of this column can beat this — and we invite you to have a go at it. The repository at github.com/seg/2016-ml-contest contains everything you need to get started, including the data and Brendon's code. We invite you to find a friend or two (or more) and have a go!

To participate, fork that repo, and add a directory for your own solution, naming it after your team. You can make pull requests with your contributions, which must be written in Python, R, or Julia. We'll run them against the blind well — the same one Brendon used in the article — and update the leaderboard. You can submit solutions as often as you like. We'll close the contest at 23:59 UT on 31 January 2017. There will be a goody bag of completely awesome and highly desirable prizes for whoever is at the top of the leaderboard when the dust settles. The full rules are in the repo.

Have fun with it, and good luck!

— Matt Hall

machine is straightforward. If the data were linearly separable, it would be easy to draw boundaries between the input data points that identified distinct classes. Figure 1 suggests that our data is not linearly separable. Using a technique known as the "kernel trick," the data is projected into a higher dimensional space where it is separable. Boundaries, or margins, can be drawn between the data in this space. These boundaries are generated during the training step.

The SVM implementation in `scikit-learn` (http://scikit-learn.org/stable/modules/svm.html) takes a number of important parameters. These can be used to control the learning rate and the specifics of the kernel functions. The choice of parameters can affect the accuracy of the classifier, and finding optimal parameter choices is an important step known as *model selection*. A succession of models is created with different parameter values, and the combination with the lowest cross -alidation error is used for the classifier. See the notebook accompanying this article for more details on the model selection procedure used to obtain the parameter choices used here.

```
>>> from sklearn import svm
>>> clf = svm.SVC(C=10, gamma=1)
```

Now we can train the classifier using the training set we created above.

```
>>> clf.fit(X_train, y_train)
```

That's it! Now that the model has been trained on our data, we can use it to predict the facies of any well with the same set of features as our training set. We set aside a well for exactly this purpose.

Figure 1. Crossplot matrix generated with the `seaborn` library.

Evaluating the classifier

To evaluate the accuracy of our classifier we will use the well we kept for a blind test and compare the predicted facies with the actual ones. We need to extract the facies labels and features of this data set and rescale the features using the same parameters used to rescale the training set.

```
>>> y_test = test_well['Facies']
>>> well_features = test_well.drop(['Facies',
                                    'Formation',
                                    'Well Name',
                                    'Depth'],
                                   axis=1)
>>> X_test = scaler.transform(well_features)
```

Now we can use our trained classifier to predict facies labels for this well, and store the results in the `Prediction` column of the `test_well` dataframe.

```
>>> y_pred = clf.predict(X_test)
>>> test_well['Prediction'] = y_pred
```

Because we know the true facies labels of the vectors in the test data set, we can use the results to evaluate the accuracy of the classifier on this well.

```
>>> from sklearn.metrics import classification_report
>>> target_names = ['SS', 'CSiS', 'FSiS', 'SiSh',
                    'MS', 'WS', 'D','PS', 'BS']
>>> print(classification_report(y_test, y_pred,
          target_names=target_names))
```

Table 3. Accuracy metrics for the test data set.

Class	precision	recall	f1-score	support
SS	0.30	0.08	0.12	89
CSiS	0.36	0.72	0.48	89
FSiS	0.62	0.54	0.58	117
SiSh	0.25	0.29	0.27	7
MS	0.17	0.11	0.13	19
WS	0.66	0.54	0.59	71
D	0.71	0.29	0.42	17
PS	0.41	0.60	0.49	40
BS	0.00	0.00	0.00	0
avg/total	0.47	0.46	0.43	449

Precision and recall are metrics that tell us how the classifier is performing for individual facies. Precision is the probability that, given a classification result for a sample, the sample actually belongs to that class. Recall is the probability that a sample will be correctly classified for a given class. For example, if the classifier predicts that an interval is fine siltstone (FSiS), there is a 62% probability that the interval is actually sandstone (precision). If an interval is actually fine siltstone, there is a 54% probability that it will be correctly classified (recall). The F1 score combines both accuracy and precision to give a single measure of relevancy of the classifier results.

Our classifier achieved an overall F1 score of 0.43 on the test well, so there is room for improvement. It is interesting to note that if we count misclassification within adjacent facies as correct, the classifier has an overall F1 score of 0.88.

Let's look at the classifier results in log-plot form. Figure 2 is based on the plots described in Alessandro Amato del Monte's excellent tutorial from June 2015 of this series. The five logs used as features are plotted along with the actual and predicted lithofacies class log.

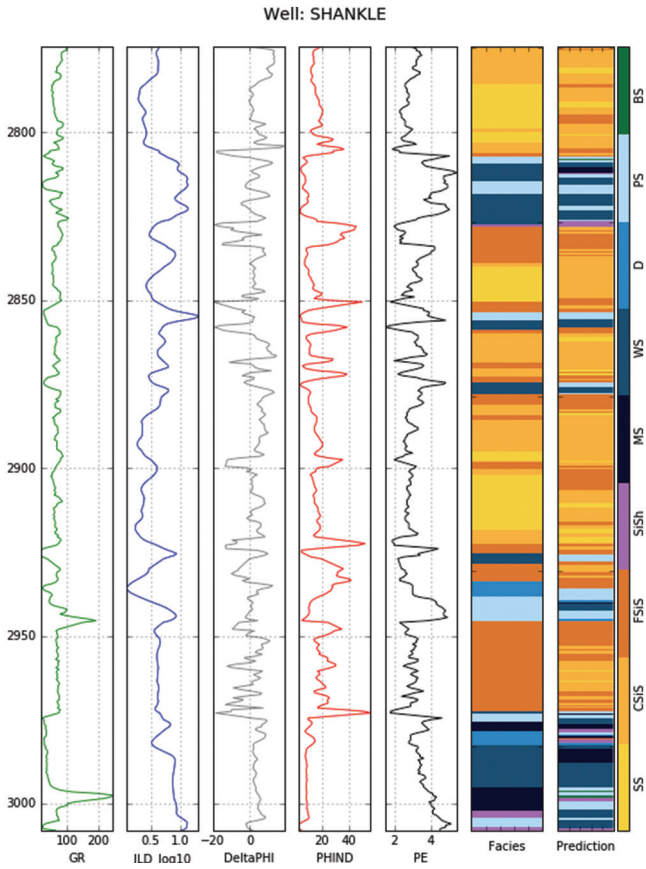

Figure 2. Well logs and facies classification results from a single well.

This tutorial has provided a brief overview of a typical machine learning workflow: preparing a data set, training a classifier, and evaluating the model. Libraries such as `scikit-learn` provide powerful algorithms that can be applied to problems in the geosciences with just a few lines of code. You can find more details as well as the data and code used for this tutorial at https://github.com/seg. TLE

References

Amato del Monte, A., 2015, Seismic petrophysics: Part 1: The Leading Edge, **34**, no. 4, 440–442, http://dx.doi.org/10.1190/tle34040440.1.

Dubois, M. K., G. C. Bohling, and S. Chakrabarti, 2007, Comparison of four approaches to a rock facies classification problem: Computers & Geosciences, **33**, no. 5, 599–617, http://dx.doi.org/10.1016/j.cageo.2006.08.011.

Waskom, M., et al., 2016, Seaborn, v0.7, January 2016: Zenodo, http://dx.doi.org/10.5281/zenodo.45133.