

Annealed Steel: PCA & Ensemble SVM

Adrian Henle & Bo Ding

```
• md"""
• # Annealed Steel: PCA & Ensemble SVM
• ### Adrian Henle & Bo Ding
• """
```

```
• using CSV, DataFrames, ScikitLearn, StatsBase, PyPlot
```




```
• md"""
• 
• """
```

```
• begin
•   @sk_import preprocessing : (OneHotEncoder, StandardScaler, label_binarize)
•   @sk_import decomposition : PCA
•   @sk_import svm: SVC
•   @sk_import metrics : (roc_auc_score, auc, roc_curve)
• end;
```


Goal

Imagine an automated factory that makes milled steel parts from iron ore. Somewhere along the line, you'll see something like the picture above: orange-hot steel rolling down a very long conveyor. That steel is undergoing a process called annealing, and it has a major effect on the ultimate working properties of the steel. If the annealing machines experience process deviations, the milling machines will need to adapt to the steel's new properties. There is no time to get a human metallurgist to test each piece of steel and determine its grade, so we would like to train an AI on simple characterization data for rapid prediction of steel quality.

- `md"""`
- `## Goal `
-
- `Imagine an automated factory that makes milled steel parts from iron ore. Somewhere along the line, you'll see something like the picture above: orange-hot steel rolling down a very long conveyor. That steel is undergoing a process called annealing, and it has a major effect on the ultimate working properties of the steel. If the annealing machines experience process deviations, the milling machines will need to adapt to the steel's new properties. There is no time to get a human metallurgist to test each piece of steel and determine its grade, so we would like to train an AI on simple characterization data for rapid prediction of steel quality.`
- `"""`

Data

The UC Irvine ML dataset **Annealing** contains a total of 868 examples of characterization data from samples of steel annealed under various conditions and labeled by grade. 100 examples are reserved for testing the model (as "operational data"). These data are split between two files, `anneal.data` and `anneal.test`, with the column names provided in `anneal.names`.

- `md"""`
- `## Data `
-
- `The UC Irvine ML dataset [Annealing] (https://archive.ics.uci.edu/ml/datasets/Annealing) contains a total of 868 examples of characterization data from samples of steel annealed under various conditions and labeled by grade. 100 examples are reserved for testing the model (as "operational data"). These data are split between two files, 'anneal.data' and 'anneal.test', with the column names provided in 'anneal.names'.`
- `"""`
-

	family	product-type	steel	carbon	hardness	temper_rolling	condition	formability
1	"?"	"C"	"A"	8	0	"?"	"S"	"?"
2	"?"	"C"	"R"	0	0	"?"	"S"	"2"

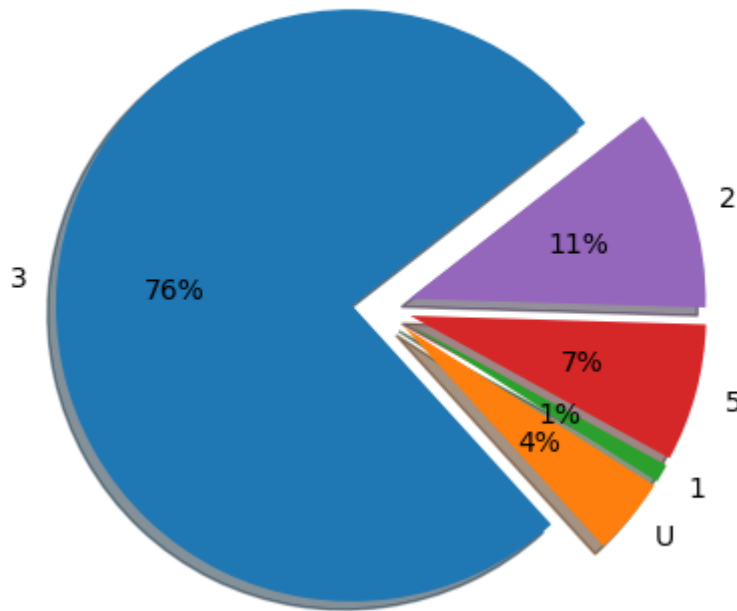
	family	product- type	steel	carbon	hardness	temper_rolling	condition	formability
3	"?"	"C"	"R"	0	0	"?"	"S"	"2"
4	"?"	"C"	"A"	0	60	"T"	"?"	"?"
5	"?"	"C"	"A"	0	60	"T"	"?"	"?"
6	"?"	"C"	"A"	0	45	"?"	"S"	"?"
7	"?"	"C"	"R"	0	0	"?"	"S"	"2"

```

• begin
•   # read data files
•   df_train = CSV.read("anneal.data", DataFrame, header=0)
•   df_test = CSV.read("anneal.test", DataFrame, header=0)
•   # anneal.names is full of info, not just column names
•   namesfile = open("anneal.names")
•   namesfiledata = readlines(namesfile)
•   close(namesfile)
•   # get ids of lines w/ names
•   attr_idx = [occursin(r"    ?[0-9]?[0-9]. [a-z]", line) for line ∈ namesfiledata]
•   names = Symbol[]
•   # loop over attribute lines
•   for line ∈ namesfiledata[attr_idx]
•       # split on whitespace, drop ""
•       nowhitespace = filter(ss -> ss ≠ "", split(line, " "))
•       # get id of token containing name
•       name_id = [occursin(r"[a-z]*-*[a-z]*:", substr) for substr ∈ nowhitespace]
•       # split name from rest of line
•       name = split(String(nowhitespace[name_id][1]), ":")[1]
•       # add name to array
•       push!(names, Symbol(name))
•   end
•   # apply column names to dataframes
•   df_train = rename(df_train, [names..., :class])
•   df_test = rename(df_test, [names..., :class])
•   df_train
• end

```

Class Label Proportions



```

• begin
•   local df = combine(g -> DataFrame(proportion=nrow(g)/nrow(df_train)),
groupby(df_train, :class))
•   clf()
•   figure()
•   title("Class Label Proportions")
•   pie(df.proportion, labels=unique(df.class), autopct="%d%%", shadow=true,
explode=0.1 .* ones(nrow(df)), startangle=38)
•   gcf()
• end

```

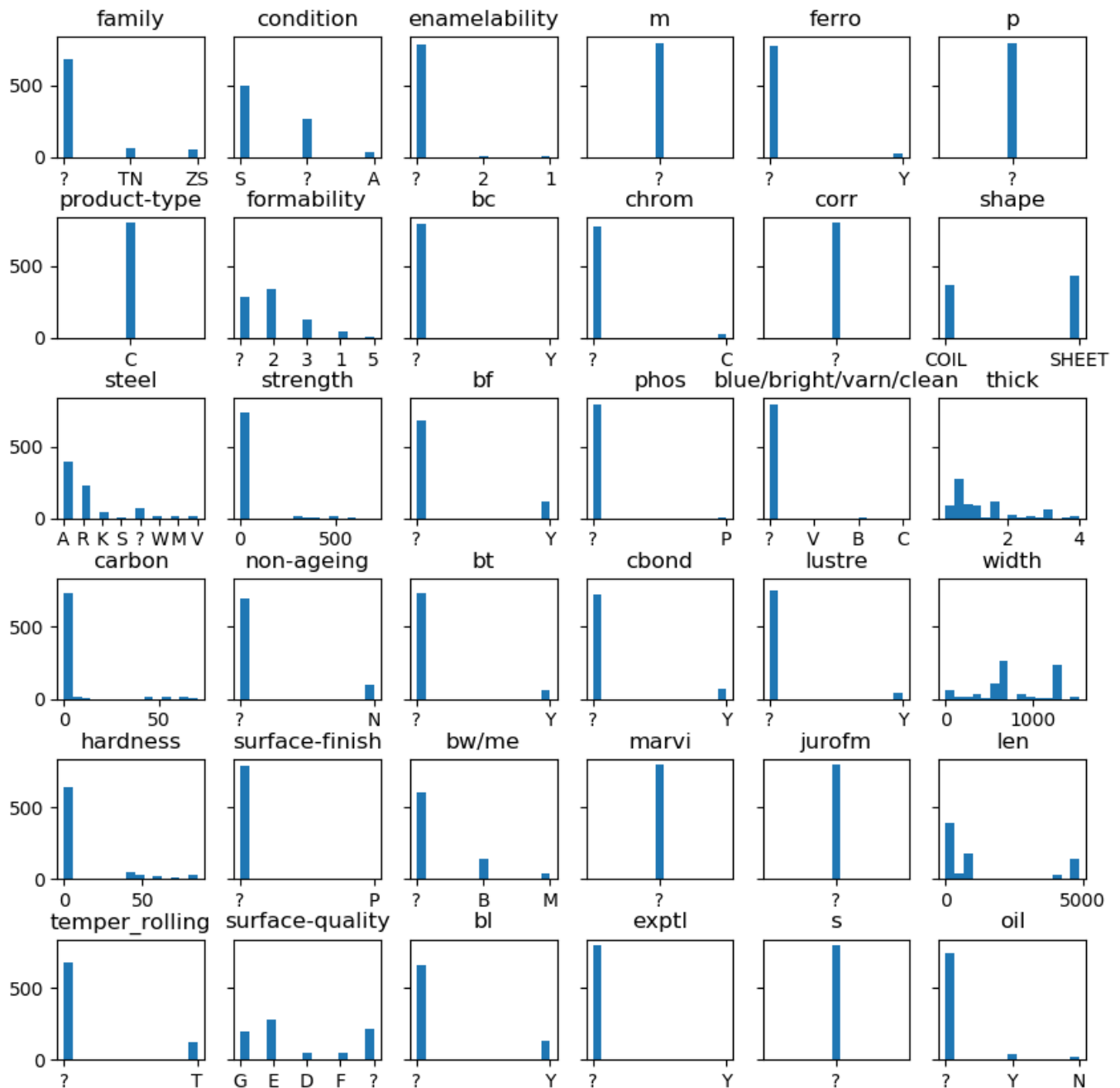
There are six grade classes (1-5 and "U") and there are 38 data features per example. Of the data features, 7 are numerical and the rest are categorical. The classes are fairly imbalanced, with class 4 totally unrepresented. Ultimately, this means that there are two caveats of this model: it will never be able to identify class 4 steel with the available training data, and we must beware of creating a classifier that simply asserts all steels are of the most common type.

Feature histograms show there are some features with no data. `anneal.names` claims these factors should have some missing values, and some "not applicable" values. The data have apparently been corrupted, and that distinction no longer exists.

```

• md"""
• There are six grade classes (1-5 and "U") and there are 38 data features per example.
Of the data features, 7 are numerical and the rest are categorical. The classes are
fairly imbalanced, with class 4 totally unrepresented. Ultimately, this means that
there are two caveats of this model: it will never be able to identify class 4 steel
with the available training data, and we must beware of creating a classifier that
simply asserts all steels are of the most common type.
•
• Feature histograms show there are some features with no data. `anneal.names` claims
these factors should have some missing values, and some "not applicable" values. The
data have apparently been corrupted, and that distinction no longer exists.

```



```

• begin
•   clf()
•   fig, axs = subplots(6, 6, sharey=True, figsize=(10,10))
•   for i = 1:(length(names)-2) # just show 36 of 38, for aesthetics
•       axs[i].hist(df_train[:, names[i]], bins=15)
•       axs[i].set_title(names[i])
•   end
•   subplots_adjust(hspace=0.5)
•   gcf()
• end

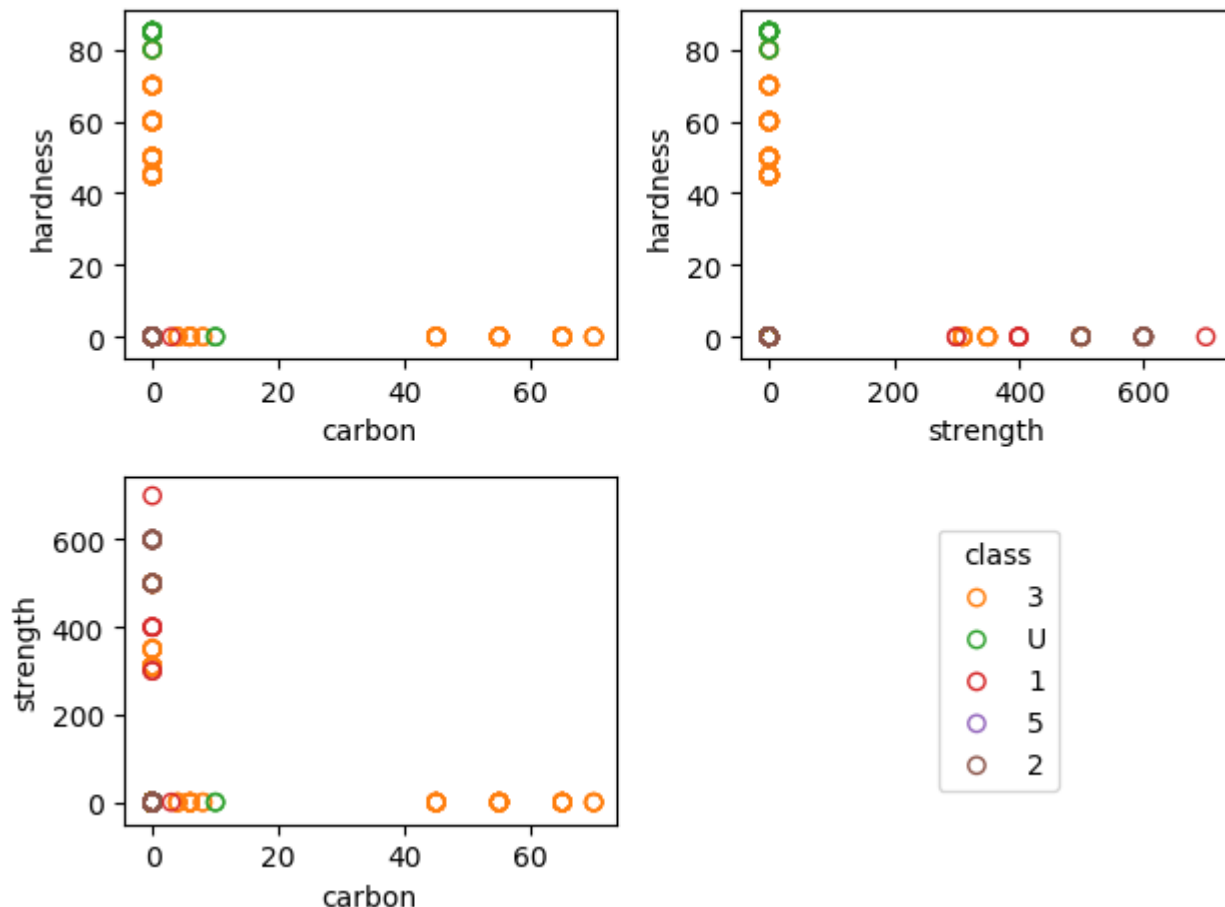
```

We can also see that our numerical variables have some decent-looking variance. Unfortunately, some of these measurements are not applicable to our scenario: we will not be getting our operational data from sample bits of metal with recordable length, width, thickness, and/or bore size. That leaves

carbon content, hardness rating, and tensile strength, all of which can be measured by non-destructive methods [1] [2] [3].

Correlation plots of numeric feature spaces:

- `md"""`
- We can also see that our numerical variables have some decent-looking variance. Unfortunately, some of these measurements are not applicable to our scenario: we will not be getting our operational data from sample bits of metal with recordable length, width, thickness, and/or bore size. That leaves carbon content, hardness rating, and tensile strength, all of which can be measured by non-destructive methods [[1\$]] (https://www.matec-conferences.org/articles/matecconf/pdf/2018/04/matecconf_nctam2018_05007.pdf) [[2\$]] ([https://www.buehler.com/nondestructive-testing.php#:~:text=Nondestructive%20testing%20\(NDT\)%20or%20Nondestructive,without%20altering%20or%20destroying%20it.&text=Typical%20types%20and%20method%20of,Rebound%2C%20and%20Ultrasonic%20Contact%20Impedance.](https://www.buehler.com/nondestructive-testing.php#:~:text=Nondestructive%20testing%20(NDT)%20or%20Nondestructive,without%20altering%20or%20destroying%20it.&text=Typical%20types%20and%20method%20of,Rebound%2C%20and%20Ultrasonic%20Contact%20Impedance.)) [[3\$]] (<https://www.bruker.com/products/x-ray-diffraction-and-elemental-analysis/handheld-xrf/applications/pmi/non-destructive-testing-ndt-xrf.html>).
-
- Correlation plots of numeric feature spaces:
- `"""`



- `# correlation plots`
- `begin`
- `feature_combos = [`
- `[:carbon, :hardness], [:carbon, :strength], [:strength, :hardness]]`
-
- `class_groups = groupby(df_train, :class)`
- `fig2, axs2 = subplots(2, 2)`
- `for (k, class) in enumerate(class_groups)`

```

•         for (l, features) ∈ enumerate(feature_combos)
•             axs2[l].scatter(class[:, features[1]], class[:, features[2]],
•                             facecolor="none", color="C$k")
•             axs2[l].set_xlabel(features[1])
•             axs2[l].set_ylabel(features[2])
•         end
•     end
•     axs2[4].axis("off")
•     fig2.legend(title = "class", labels=[k[1] for k ∈ keys(class_groups)],
•         loc=(0.75, 0.15))
•     tight_layout()
•    (gcf())
• end

```

There is some separation along the axes, but no example had two of the three measurements. Training a model to predict on the numeric columns alone may not lead to sufficiently accurate decision functions. The number of multi-label categorical features makes visualizing relationships cumbersome. To simplify the problem, we reduce the number of dimensions using PCA.

```

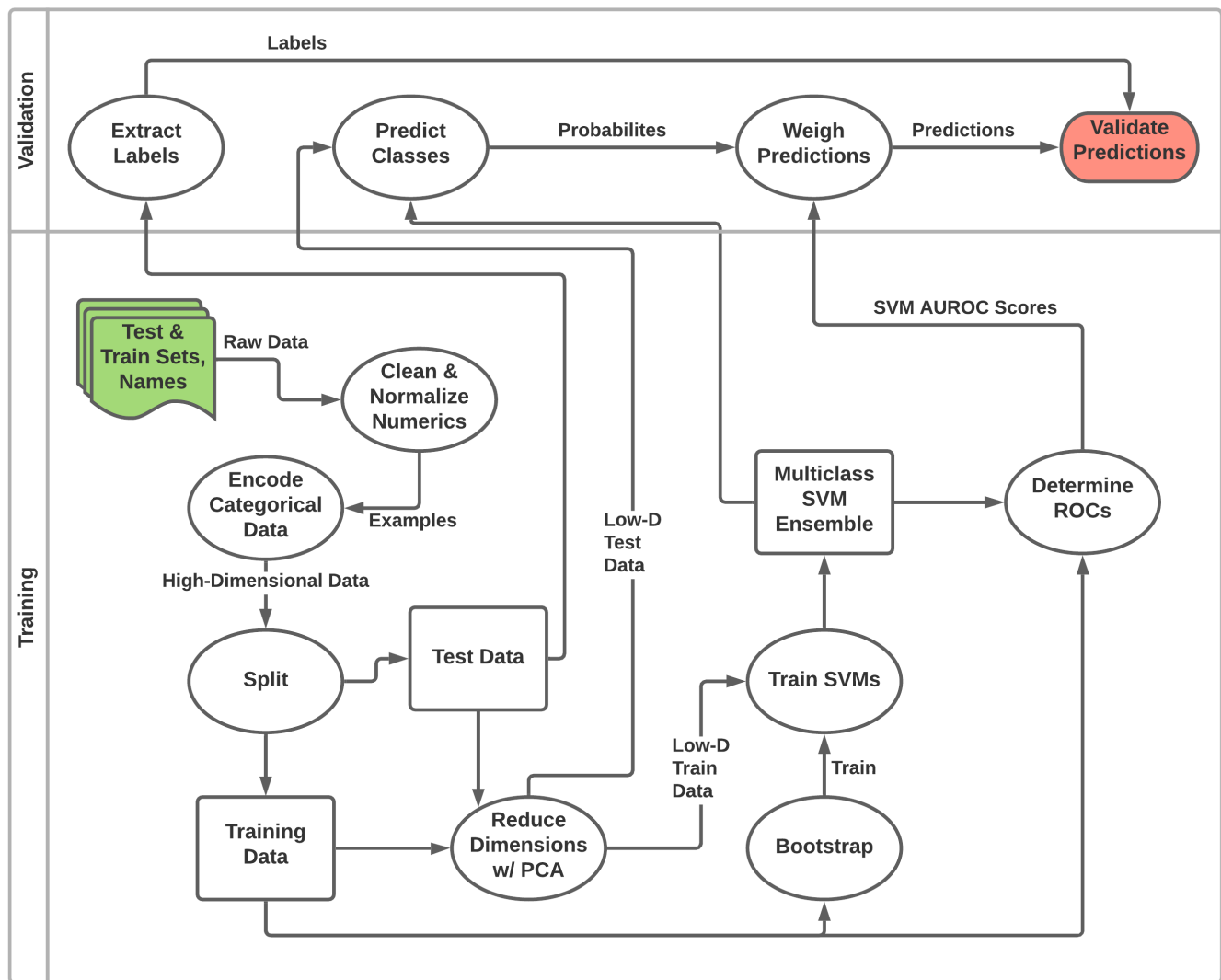
• md"""
• There is some separation along the axes, but no example had two of the three
• measurements. Training a model to predict on the numeric columns alone may not lead
• to sufficiently accurate decision functions. The number of multi-label categorical
• features makes visualizing relationships cumbersome. To simplify the problem, we
• reduce the number of dimensions using PCA.
• """

```

Approach

To render the problem more approachable, we decided to reduce the dimensionality of the input data via Principal Component Analysis (PCA). After dimensional reduction, we train an ensemble of Support Vector Machines (SVMs) to perform multi-class classification. The rationale for using an ensemble instead of a single SVM for classification is the small number of training data; the weak learners of the ensemble will help capture more variance of the original data set without overfitting.

Here is the Data Flow Diagram describing the ML architecture we employed:



- md"""
- ## Approach
-
- To render the problem more approachable, we decided to reduce the dimensionality of the input data via Principal Component Analysis (PCA). After dimensional reduction, we train an ensemble of Support Vector Machines (SVMs) to perform multi-class classification. The rationale for using an ensemble instead of a single SVM for classification is the small number of training data; the weak learners of the ensemble will help capture more variance of the original data set without overfitting.
-
- Here is the Data Flow Diagram describing the ML architecture we employed:
-
- ![DFD](https://github.com/eahenle/CHE599-Project/raw/main/CHE599_DFD.png)
- """

Data Preparation

Categorical data must be cast to pseudo-numeric for PCA. We used one-hot-drop-one encoding, meaning that each categorical variable with n represented categories will be replaced by $n - 1$ Boolean columns indicating whether or not the example belongs to any given category.

The String-type columns of data are the only ones that should be encoded. Column 39 is the classification label, so it is also excluded from encoding. The net result is an addition of 5 feature columns.

- `md"""`
- `## Data Preparation`
- `Categorical data must be cast to pseudo-numeric for PCA. We used one-hot-drop-one encoding, meaning that each categorical variable with n represented categories will be replaced by $n-1$ Boolean columns indicating whether or not the example belongs to any given category.`
- `The 'String'-type columns of data are the only ones that should be encoded. Column 39 is the classification label, so it is also excluded from encoding. The net result is an addition of 5 feature columns.`
- `"""`

```
str_cols =
  Symbol[:family, Symbol("product-type"), :steel, :temper_rolling, :condition, :formabi
```

- `# id the String-type columns`
- `str_cols = [names[i] for (i,col) ∈ enumerate(eachcol(df_train)) if`
- `typeof(col[1]) == String && i < 39]`

	xo_TN	xo_ZS	x2_A	x2_K	x2_M	x2_R	x2_S	x2_V	mor
1	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	
2	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	
3	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	
4	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	
5	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	
6	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
8	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	
9	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	
10	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	
more									

- `begin`
- `# prepare the one-hot-drop-one encoder`
- `one_hot_encoder = OneHotEncoder(drop="first", sparse=false)`
- `# do the encoding`
- `train_encoded = one_hot_encoder.fit_transform(convert(Matrix, df_train[:,`
- `str_cols]))`
- `# put train_encoded back into DF form, attach col names`
- `train_encoded = convert(DataFrame, train_encoded)`
- `train_encoded = rename(train_encoded, [convert(Symbol, name) for name ∈`
- `one_hot_encoder.get_feature_names()])`

```

• test_encoded = one_hot_encoder.transform(convert(Matrix, df_test[:, str_cols]))
• test_encoded = convert(DataFrame, test_encoded)
• test_encoded = rename(test_encoded, [convert(Symbol, name) for name ∈
one_hot_encoder.get_feature_names()])
• end

```

The numerical columns need their own pre-processing for PCA: normalization.

```

• md"""
• The numerical columns need their own pre-processing for PCA: normalization.
• """

```

	carbon	hardness	strength	thick	width	len	bore
1	-0.261159	1.3514	-0.267167	0.485621	-0.424617	-0.675142	-0.226046
2	-0.261159	-0.477458	-0.267167	-0.560754	-0.424864	-0.675142	-0.226046
3	-0.261159	2.97704	-0.267167	-0.907997	-0.424617	-0.271023	-0.226046
4	-0.261159	1.5546	-0.267167	-0.907997	-0.424617	-0.675142	-0.226046
5	-0.261159	-0.477458	-0.267167	-0.560754	1.33077	-0.675142	-0.226046
6	-0.261159	-0.477458	-0.267167	-0.443458	-1.80915	-0.675142	-0.226046
7	-0.261159	-0.477458	4.06263	0.485621	-1.19081	-0.51551	-0.226046
8	-0.261159	-0.477458	-0.267167	-0.443458	-0.424617	-0.271023	-0.226046
9	-0.261159	-0.477458	-0.267167	0.485621	-0.424617	1.53637	-0.226046
10	-0.261159	-0.477458	-0.267167	0.0210818	1.15771	-0.271023	-0.226046
more							

```

100 -0.261159 2.97704 -0.267167 3.27286 -0.424617 -0.675142 3.78251
• begin
• num_cols = [col for col ∈ names if !(col ∈ str_cols)]
• numerics_train = convert(Matrix, df_train[:, num_cols])
• stdscaler = StandardScaler()
• numerics_train = convert(DataFrame, stdscaler.fit_transform(numerics_train))
• numerics_train = rename(numerics_train, num_cols)
• numerics_test = convert(Matrix, df_test[:, num_cols])
• numerics_test = convert(DataFrame, stdscaler.transform(numerics_test))
• numerics_test = rename(numerics_test, num_cols)
• end

```

Now we can recombine the numerical and encoded data. It is also a good time to drop the non-relevant physical features.

```

• md"""
• Now we can recombine the numerical and encoded data. It is also a good time to drop
the non-relevant physical features.
• """

```

	carbon	hardness	strength	xo_TN	xo_ZS	x2_K	x2_M	x2_R
1	0.327764	-0.477458	-0.267167	0.0	0.0	0.0	0.0	0.0
2	-0.261159	-0.477458	-0.267167	0.0	0.0	0.0	0.0	1.0
3	-0.261159	-0.477458	-0.267167	0.0	0.0	0.0	0.0	1.0
4	-0.261159	1.96101	-0.267167	0.0	0.0	0.0	0.0	0.0
5	-0.261159	1.96101	-0.267167	0.0	0.0	0.0	0.0	0.0
6	-0.261159	1.3514	-0.267167	0.0	0.0	0.0	0.0	0.0
7	-0.261159	-0.477458	-0.267167	0.0	0.0	0.0	0.0	1.0
8	-0.261159	-0.477458	-0.267167	0.0	0.0	0.0	0.0	0.0
9	-0.261159	-0.477458	-0.267167	0.0	0.0	0.0	0.0	1.0

```

• begin
•   # combine pre-processed data
•   clean_training_data = hcat(numerics_train, train_encoded)
•   clean_test_data = hcat(numerics_test, test_encoded)
•   # drop length, width, thickness, and bore size data
•   for df ∈ [clean_training_data, clean_test_data]
•       for col ∈ drop_cols
•           select!(df, Not(col))
•       end
•   end
•   clean_training_data
• end

```

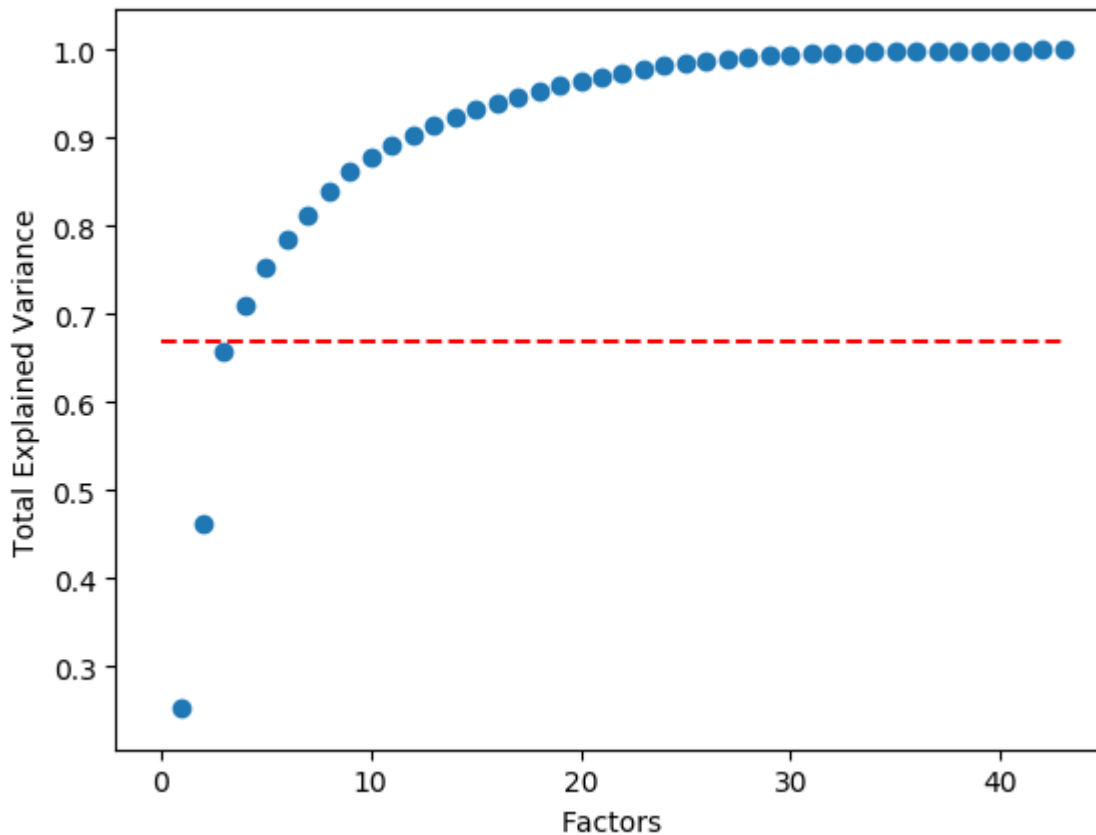
Principal Component Analysis

PCA is performed with `sklearn` to reduce the variable space to only as many dimensions as needed. After learning the principal component vectors of the feature space, the training and test data must be transformed into principal component space (PC-space), as approximated by the first n PC vectors. The total explained variance is plotted for the number of factors used:

```

• md"""
• ## Principal Component Analysis
•
• PCA is performed with `sklearn` to reduce the variable space to only as many
• dimensions as needed. After learning the principal component vectors of the feature
• space, the training and test data must be transformed into principal component space
• (PC-space), as approximated by the first $n$ PC vectors. The total explained variance
• is plotted for the number of factors used:
• """

```



```

• begin
•   # create the PCA decomposer and learn PC space
•   pca = PCA()
•   pca.fit(convert(Matrix, clean_training_data))
•
•   figure()
•   varsums = [sum(pca.explained_variance_ratio_[1:i]) for i in 1:pca.n_components_]
•   scatter(1:pca.n_components_, varsums)
•   xlabel("Factors")
•   ylabel("Total Explained Variance")
•   plot([0, pca.n_components_], [pca_var_thresh, pca_var_thresh], color="red",
•        linestyle="--")
•
•   pca = PCA(n_components=pca_var_thresh)
•   pca.fit(convert(Matrix, clean_training_data))
•   transformed_training_data = pca.transform(convert(Matrix, clean_training_data))
•   transformed_test_data = pca.transform(convert(Matrix, clean_test_data))
•
•   gcf()
• end

```

With the threshold at 67% and the principal components selected, the data are transformed into PC-space. Now instead of 43 dimensions, our data have only 4.

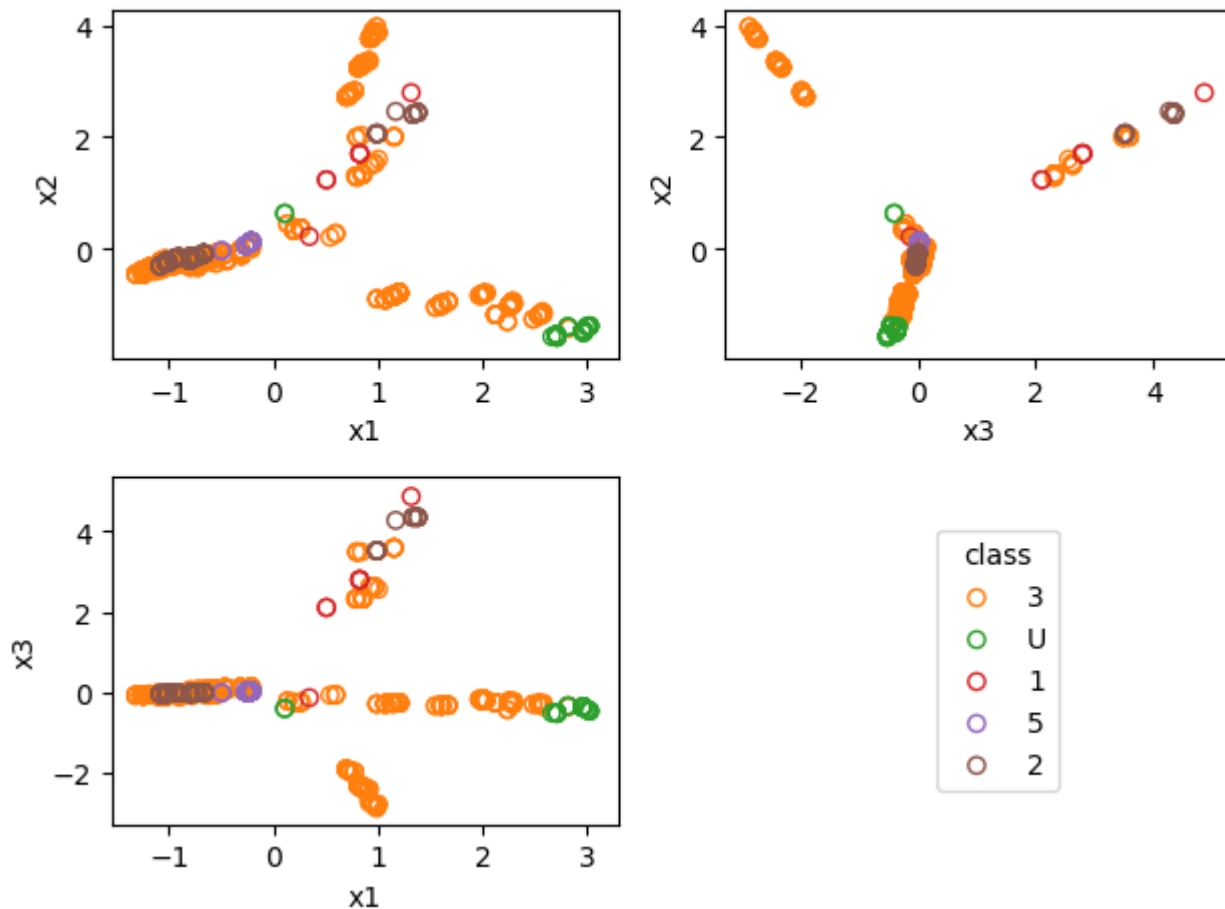
We can visualize the class distributions in the hyperplanes defined by the first three principal component vectors.

```

• md"""
• With the threshold at $(Int(round(pca_var_thresh*100)))% and the principal components
• selected, the data are transformed into PC-space. Now instead of
• $(ncol(clean_training_data)) dimensions, our data have only $(pca.n_components_).
•

```

- We can visualize the class distributions in the hyperplanes defined by the first three principal component vectors.
- """



```

• # visualize distributions in hyperplanes of first three PCs
• begin
•     pc_combos = [[:x1, :x2], [:x1, :x3], [:x3, :x2]]
•     df_transformed_training_data = convert(DataFrame, transformed_training_data)
•     df_transformed_training_data[!, :class] = df_train[:, :class]
•     transformed_class_groups = groupby(df_transformed_training_data, :class)
•     fig3, axs3 = subplots(2, 2)
•     for (k, class) in enumerate(transformed_class_groups)
•         for (l, features) in enumerate(pc_combos)
•             axs3[l].scatter(class[:, features[1]], class[:, features[2]],
• facecolor="none", color="C$k")
•             axs3[l].set_xlabel(features[1])
•             axs3[l].set_ylabel(features[2])
•         end
•     end
•     axs3[4].axis("off")
•     fig3.legend(title = "class", labels=[k[1] for k in keys(class_groups)], loc=(0.75,
0.15))
•     tight_layout()
•     gcf()
• end

```

We can start to see that the data have been made much more separable. The features from the original data which have the largest contributions are:

```

• md"""

```

- We can start to see that the data have been made much more separable. The features from the original data which have the largest contributions are:
- """

```
String["hardness", "carbon", "strength", "x0_TN"]
```

- begin
- id = [m[2] for m in findmax([pca.components_[i,:] for i in 1:pca.n_components_])]
- DataFrames.names(clean_training_data)[id]
- end

The hardness, carbon content, and tensile strength are the most important features of the first 3 principal components. They are not the only important features, but they make the largest contributions. Intuitively, this is good—the grade of steel is naturally dependent on these features (a fact that the PCA determined for itself!).

The most important feature for the fourth principal component is "x0_TN", meaning whether or not the steel is in a particular family defined by its chemical makeup.

Support Vector Machine Classification

SVM can perform multi-class classification by learning multiple decision boundaries within a multi-dimensional space (in this case, PC-space). We chose to use an ensemble of SVMs trained on bootstrapped data to capture additional variance beyond what a single learner can offer.

- md"""
- The hardness, carbon content, and tensile strength are the most important features of the first 3 principal components. They are not the only important features, but they make the largest contributions. Intuitively, this is good—the grade of steel is naturally dependent on these features (a fact that the PCA determined for itself!).
-
- The most important feature for the fourth principal component is "x0_TN", meaning whether or not the steel is in a particular family defined by its chemical makeup.
-
- ## Support Vector Machine Classification
- SVM can perform multi-class classification by learning multiple decision boundaries within a multi-dimensional space (in this case, PC-space). We chose to use an ensemble of SVMs trained on bootstrapped data to capture additional variance beyond what a single learner can offer.
- """

- begin
- *# create ensemble*
- svc_ensemble = [SVC(decision_function_shape="ovr", break_ties=true,
- random_state=0, probability=true) for i in 1:nb_SVMs]
- *# bootstrap data*
- bootstrap_samples = []
- bootstrap_sample_labels = []
- for _ in 1:nb_SVMs
- ids = StatsBase.sample(1:nrow(clean_training_data),
- nrow(clean_training_data), replace=true)
- *# store bootstrapped data*
- push!(bootstrap_samples, transformed_training_data[ids, :])
- push!(bootstrap_sample_labels, df_train[ids, :class])
- end

```

• # train SVMs
• for (i, svc) ∈ enumerate(svc_ensemble)
•     svc.fit(convert(Matrix, bootstrap_samples[i]), bootstrap_sample_labels[i])
• end
• end

```

To assess the predictive quality of each SVM, the Area Under the Receiver Operating Characteristic Curve (AUROC) is calculated for each learner by comparing it against the original (full) training data. This should be done slightly differently (each learner should be internally validated against the training data not included in its bootstrapped set), but this approach was chosen for expedience. As such, these are not "true" AUROCs; they are handy approximations.

```

• begin
•     aurocs = []
•     # run predict_proba on all examples for each SVM
•     svc_probs = [
•         svc_ensemble[i].predict_proba(transformed_training_data) for i ∈ 1:nb_SVMs]
•     # avg over SVMs by class for each example (n_samples, n_classes)
•     class_probabilities = zeros(nrow(df_train), length(unique(df_train[:, :class])))
•     # loop over SVMs
•     for mat ∈ svc_probs
•         # one ROC per mat (SVM)
•         push!(aurocs, roc_auc_score(df_train[:, :class], mat, multi_class="ovr"))
•         # loop over classes
•         for c ∈ 1:length(unique(df_train[:, :class]))
•             # loop over examples
•             for e ∈ 1:nrow(df_train)
•                 # accumulate probabilities
•                 class_probabilities[e, c] += mat[e, c]
•             end
•         end
•     end
•     class_probabilities ./= nb_SVMs
• end;

```

The probabilities for each class according to the ensemble are calculated on every example, and summing these over the training data returns a very good approximation of the original number of examples of each class.

```

• md"""
• The probabilities for each class according to the ensemble are calculated on every
• example, and summing these over the training data returns a very good approximation of
• the original number of examples of each class.
• """

```

```
Float64[7.7, 79.5, 616.8, 59.4, 34.5]
```

```
• round.(sum.(eachcol(class_probabilities)), digits=1)
```

The AUROC of the ensemble for the training data is quite good: 0.991

The AUROC score of each individual learner is high. However, few if any of these SVMs will be able to match the performance of the ensemble.

```

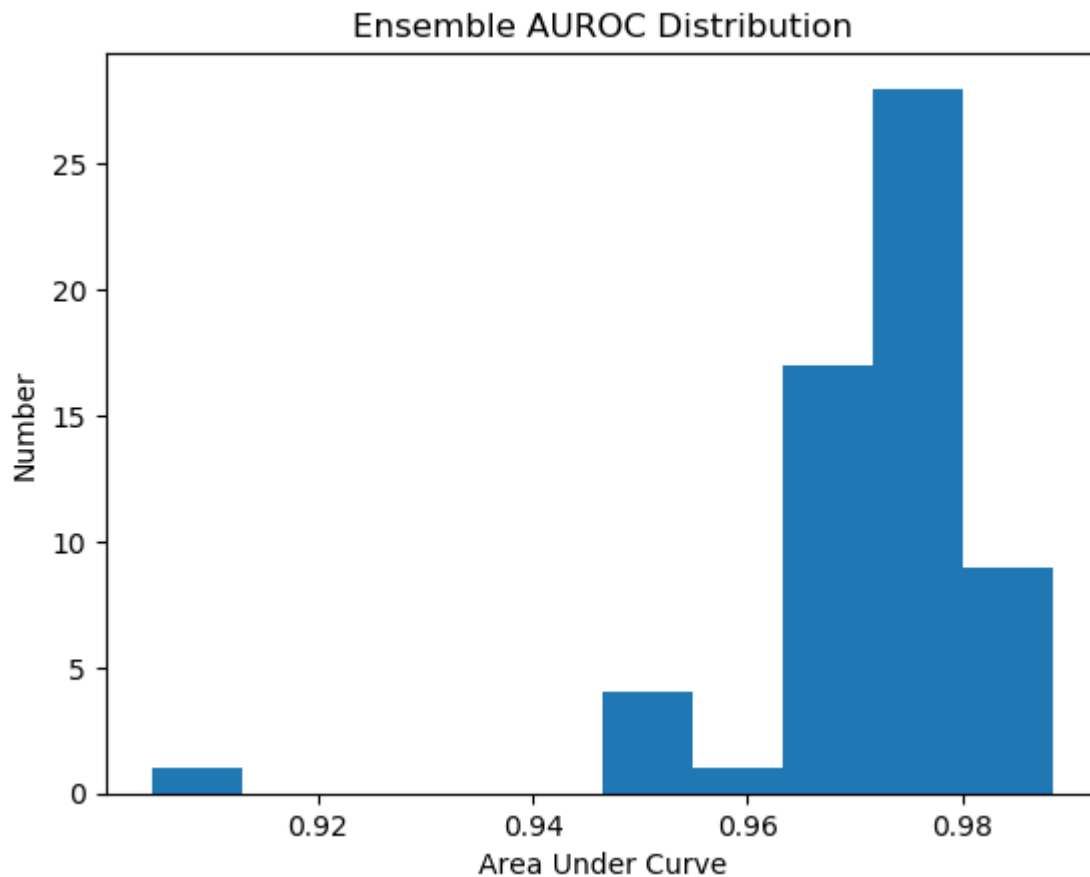
• md"""
• The AUROC of the ensemble for the training data is quite good:
• $(round(roc_auc_score(df_train[:, :class], class_probabilities, multi_class="ovr"),

```



```
digits=3))
```

- The AUROC score of each individual learner is high. However, few if any of these SVMs will be able to match the performance of the ensemble.
- """



- *# make this a histogram*
- begin
- `figure()`
- `hist(aurocs)`
- `title("Ensemble AUROC Distribution")`
- `xlabel("Area Under Curve")`
- `ylabel("Number")`
- `gcf()`
- end

Model Validation on Operating Data

The time has come to evaluate the model on our reserved testing data. First, an array of probability matrices is collected, with one matrix for each SVM, containing the class probability estimates for each class on each test example.

- `md"""`
- `## Model Validation on Operating Data`
-
- The time has come to evaluate the model on our reserved testing data. First, an array of probability matrices is collected, with one matrix for each SVM, containing the class probability estimates for each class on each test example.
- `"""`

```

• begin
•     prob_mats = []
•     # loop over SVMs
•     for i ∈ 1:nb_SVMs
•         probsi = zeros(nrow(df_test), length(unique(df_train.class)))
•         # loop over examples
•         for j ∈ 1:nrow(df_test)
•             # get weighted predictions from each SVM for each test example
•             probsi[j,:] = svc_ensemble[i].predict_proba(
•                 [convert(Array, transformed_test_data[j,:])]
•             )
•         end
•         push!(prob_mats, probsi)
•     end
• end

```

The probabilities are attenuated by multiplying them against the SVMs' normalized training AUROC scores, and accumulating the sums so that we end up with an array of 5 weighted class scores for each test example. The largest score decides the classification.

```

• md"""
• The probabilities are attenuated by multiplying them against the SVMs' normalized
• training AUROC scores, and accumulating the sums so that we end up with an array of 5
• weighted class scores for each test example. The largest score decides the
• classification.
• """

```

```

• begin
•     # loop over SVM probability matrices
•     for i ∈ 1:nb_SVMs
•         # multiply each value in matrix i in prob_mats by svm_score[i]
•         prob_mats[i] .*= aurocs[i] - 0.5*auroc_delta
•     end
•     # one array of 5 score sums for each validation example
•     sums = zeros(nrow(df_test), length(unique(df_train.class)))
•     for i ∈ 1:nb_SVMs
•         for j ∈ 1:nrow(df_test)
•             for k ∈ 1:length(unique(df_train.class))
•                 sums[j, k] += prob_mats[i][j, k]
•             end
•         end
•     end
•     probs = sums ./ sum.([sums[i, :] for i in 1:size(sums, 1)])
• end;

```

	predicted	actual	correct
1	"3"	"3"	true
2	"3"	"3"	true
3	"0"	"0"	true
4	"3"	"3"	true
5	"3"	"3"	true
6	"5"	"5"	true

	predicted	actual	correct
7	"2"	"2"	true
8	"3"	"2"	false
9	"3"	"3"	true

```

• begin
•   final_predictions =
•     [svc_ensemble[1].classes_[t[2]] for t in findmax.(eachrow(sums))]
•   result_df = DataFrame(predicted=final_predictions, actual=df_test.class,
•     correct=final_predictions .== df_test.class)
• end

```

The ensemble mis-classifies 10 of 100 test cases.

```

• md"""
• The ensemble mis-classifies $(count(c -> c == false, result_df.correct)) of
• $(nrow(df_test)) test cases.
• """

```

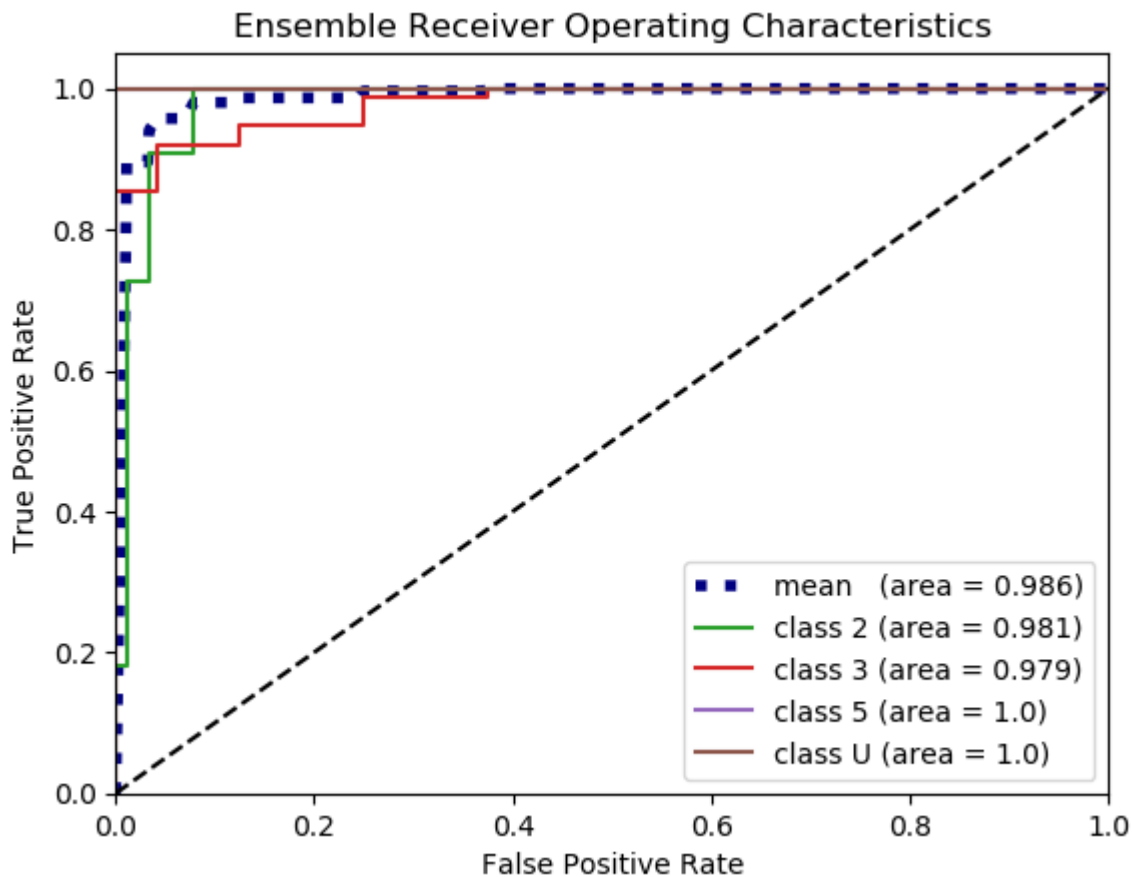
ROC Curves for Ensemble

The ensemble's ROC curve is determined for each class represented in the test data. There are no examples of class 1 steel in the test set, so its ROC cannot be calculated. Ultimately, the ensemble model's performance on the test data is roughly as good as the ensemble AUROC previously estimated using the training data, and the model does a fairly good job of predicting the grade of steel based on metrics available to in-line monitoring systems.

```

• md"""
• ## ROC Curves for Ensemble
•
• The ensemble's ROC curve is determined for each class represented in the test data.
• There are no examples of class 1 steel in the test set, so its ROC cannot be
• calculated. Ultimately, the ensemble model's performance on the test data is roughly
• as good as the ensemble AUROC previously estimated using the training data, and the
• model does a fairly good job of predicting the grade of steel based on metrics
• available to in-line monitoring systems.
• """

```



```

begin
  # because I couldn't get scipy to interpolate things for me
  function get_tpr(all_fpr, fpr_i, tpr_i)
    all_tpr = zeros(length(all_fpr))
    for (i, fpr) in enumerate(all_fpr)
      idx = findlast(e -> e ≤ fpr, fpr_i)
      if isnothing(idx)
        idx = 1
      end
      all_tpr[i] = tpr_i[idx]
    end
    return all_tpr
  end

  # the rest of this cell is a modified translation of the examples found at
  # https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html
  y_test = label_binarize(df_test.class,
    classes=["$(c[1])" for c in svc_ensemble[1].classes_])
  n_classes = 5
  fpr = [Float64[] for _ ∈ 1:5]
  tpr = [Float64[] for _ ∈ 1:5]
  roc_auc = zeros(5)
  for i ∈ 1:n_classes
    fpr[i], tpr[i], _ = roc_curve(y_test[:, i], probs[:, i])
    tpr[i] = isnan(tpr[i][1]) ? zeros(length(tpr[i])) : tpr[i]
    roc_auc[i] = auc(fpr[i], tpr[i])
  end

  all_fpr = []
  for sub_arr ∈ fpr
    for e ∈ sub_arr
      push!(all_fpr, e)
    end
  end

  all_fpr = sort(unique(all_fpr))
  mean_tpr = [zeros(length(all_fpr))]

```

```

•   for i in 1:n_classes
•       mean_tpr[1] += get_tpr(all_fpr, fpr[i], tpr[i])
•   end
•   mean_tpr = mean_tpr[1]
•   mean_tpr /= n_classes - 1
•   mean_tpr[1] = 0
•   mean_tpr[end] = 1
•   figure()
•   plot(all_fpr, mean_tpr,
•       label="mean (area = $(round(auc(all_fpr, mean_tpr), digits=3)))",
•       color="navy", linestyle=":", linewidth=4)
•   for i in 2:n_classes # class "1" isn't in the testing set
•       class = svc_ensemble[1].classes_[i]
•       area = round(roc_auc[i], digits=3)
•       plot(fpr[i], tpr[i], color="C$i",
•           label="class $class (area = $area)")
•   end
•   plot([0, 1], [0, 1], "k--")
•   xlim([0.0, 1.0])
•   ylim([0.0, 1.05])
•   xlabel("False Positive Rate")
•   ylabel("True Positive Rate")
•   title("Ensemble Receiver Operating Characteristics")
•   legend(loc="lower right")
•   gcf()
• end

```

```

• begin
•   # some of these factors were discovered to be unavailable after ensemble analysis
•   drop_cols = [:bore, :len, :width, :thick, :x6_N, :x2_A, :x5_3, :x28_SHEET]
•   # pca variance threshold
•   pca_var_thresh = 0.67 # 0.67
•   # number of SVMs in the ensemble
•   nb_SVMs = 60
•   # enhanced auROC attenuation (not)
•   auROC_delta = false
• end;

```