Linear Models

PHYS 453

Dr Daugherity

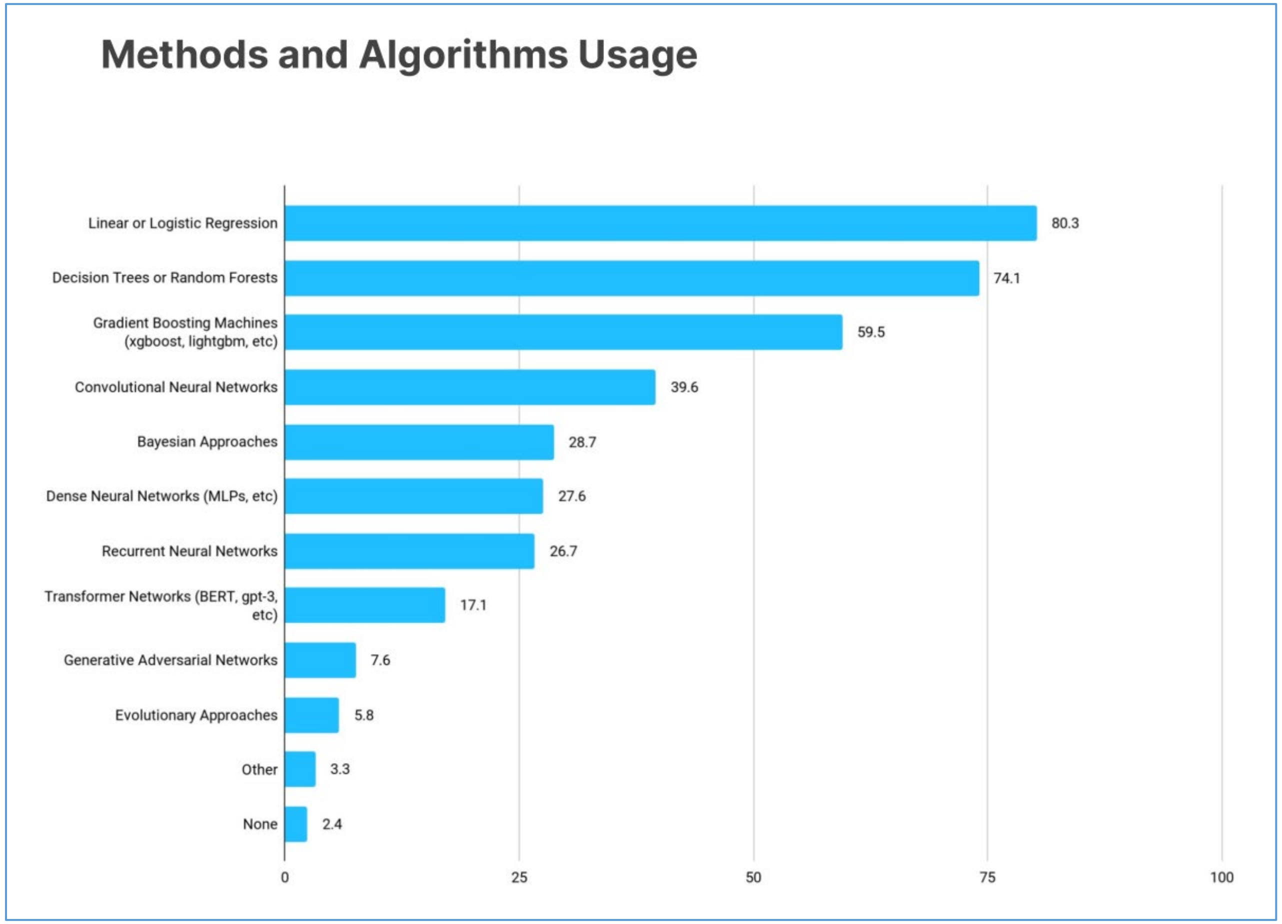
ACU
ABILENE CHRISTIAN UNIVERSITY

Road to Neural Networks

- This begins our Road to **Neural Networks**
- A Neural Network is simply a bunch of linear models combined in non-linear ways

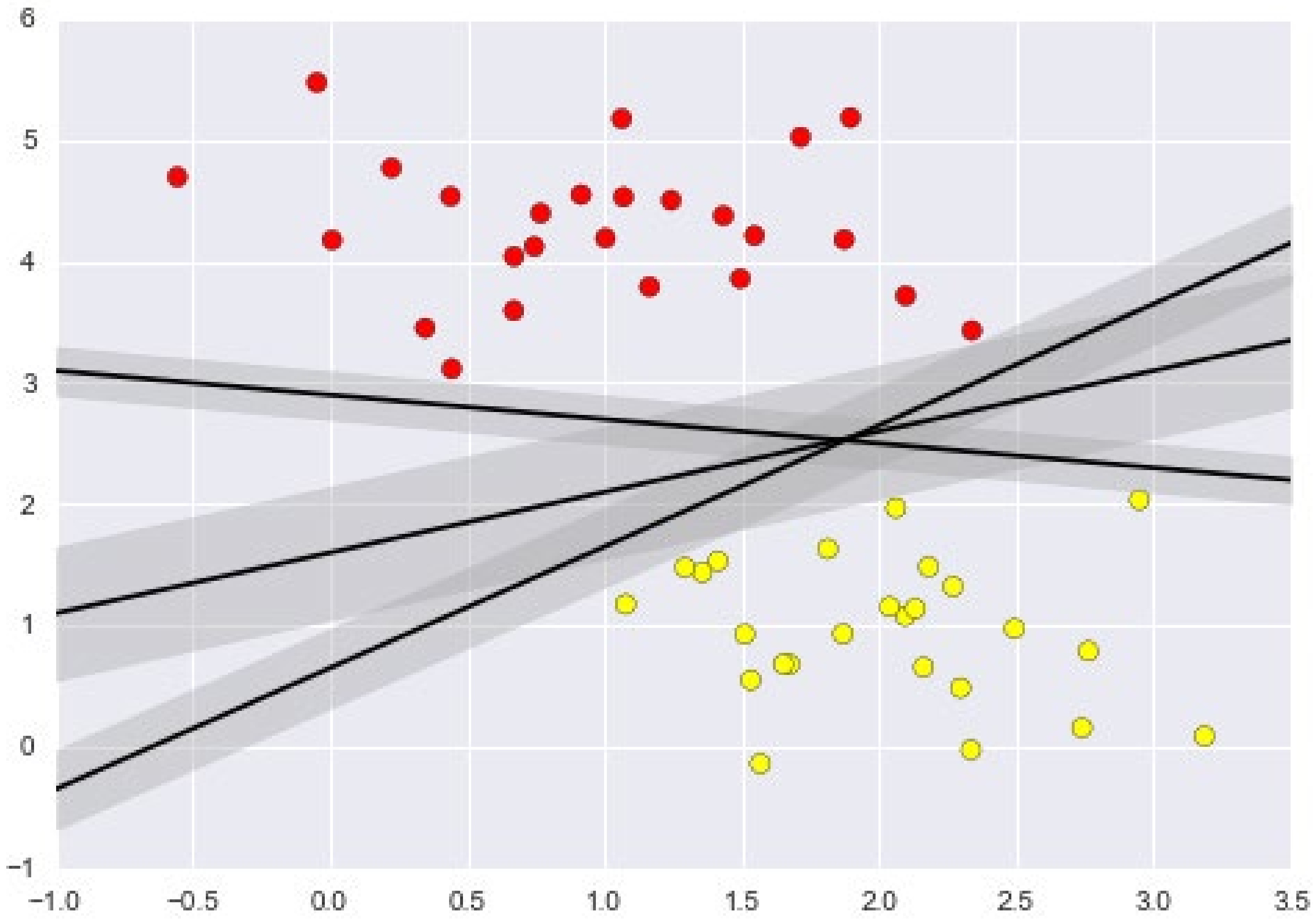
Linear Models

- Many seem overly simplistic now, but often surprisingly effective and a good first thing to try
- Basis for neural networks!
- Insights into feature importance!
- Clever ways to extend with non-linear kernels



https://www.kaggle.com/kaggle-survey-2021

Problem Statement: For a linear classifier we need to find the line which best separates the targets



All 3 lines work. Which is best?

Mathematically, we expect to have either zero or an infinite number of solutions.

https://jakevdp.github.io/PythonDataScienceHandbook/05.07-support-vector-machines.html

Linear Models

Find the "best" straight line that separates the data.

Different approaches:

- Fit ("Regression") – use Linear algebra tricks to find the least squares fit. Deterministic and robust.
- Margin (SVM) – focus only on the data points near the line and maximize the *margin* from randomized start.
- Search (SGD) – define a generic *loss function* and find that minimizes it with randomized (stochastic) search

WARNING: This gets complicated. We'll be simplifying a lot.

https://scikit-learn.org/stable/supervised_learning.html

REGRESSION *(mostly)*

1.1. Linear Models

- 1.1.1. Ordinary Least Squares
- 1.1.2. Ridge regression and classification
- 1.1.3. Lasso
- 1.1.4. Multi-task Lasso
- 1.1.5. Elastic-Net
- 1.1.6. Multi-task Elastic-Net
- 1.1.7. Least Angle Regression
- 1.1.8. LARS Lasso
- 1.1.9. Orthogonal Matching Pursuit (OMP)
- 1.1.10. Bayesian Regression
- 1.1.11. Logistic regression
- 1.1.12. Stochastic Gradient Descent - SGD
- 1.1.13. Perceptron
- 1.1.14. Passive Aggressive Algorithms
- 1.1.15. Robustness regression: outliers and modeling errors
- 1.1.16. Polynomial regression: extending linear models with basis functions

MARGIN

1.4. Support Vector Machines

- 1.4.1. Classification
- 1.4.2. Regression
- 1.4.3. Density estimation, novelty detection
- 1.4.4. Complexity
- 1.4.5. Tips on Practical Use
- 1.4.6. Kernel functions
- 1.4.7. Mathematical formulation
- 1.4.8. Implementation details

SEARCH

1.5. Stochastic Gradient Descent

- 1.5.1. Classification
- 1.5.2. Regression
- 1.5.3. Stochastic Gradient Descent for sparse data
- 1.5.4. Complexity
- 1.5.5. Stopping criterion
- 1.5.6. Tips on Practical Use
- 1.5.7. Mathematical formulation
- 1.5.8. Implementation details

Regression Models

https://scikit-learn.org/stable/modules/linear_model.html

1.1.2.2. Classification

The `Ridge` regressor has a classifier variant: `RidgeClassifier`. This classifier first converts binary targets to `{-1, 1}` and then treats the problem as a regression task, optimizing the same objective as above. The predicted class corresponds to the sign of the regressor's prediction. For multiclass classification, the problem is treated as multi-output regression, and the predicted class corresponds to the output with the highest value.

1.1.2.1. Regression

`Ridge` regression addresses some of the problems of Ordinary Least Squares by imposing a penalty on the size of the coefficients. The ridge coefficients minimize a penalized residual sum of squares:

$$\min_{w} ||Xw - y||_2^2 + \alpha ||w||_2^2$$

The complexity parameter $\alpha \geq 0$ controls the amount of shrinkage: the larger the value of α, the greater the amount of shrinkage and thus the coefficients become more robust to collinearity.

sklearn.linear_model.RidgeClassifier

class `sklearn.linear_model.`**`RidgeClassifier`**(*alpha=1.0, *, fit_intercept=True, copy_X=True, max_iter=None, tol=0.0001, class_weight=None, solver='auto', positive=False, random_state=None*) [source]

Classifier using Ridge regression.

This classifier first converts the target values into `{-1, 1}` and then treats the problem as a regression task (multi-output regression in the multiclass case).

Read more in the User Guide.

Parameters:

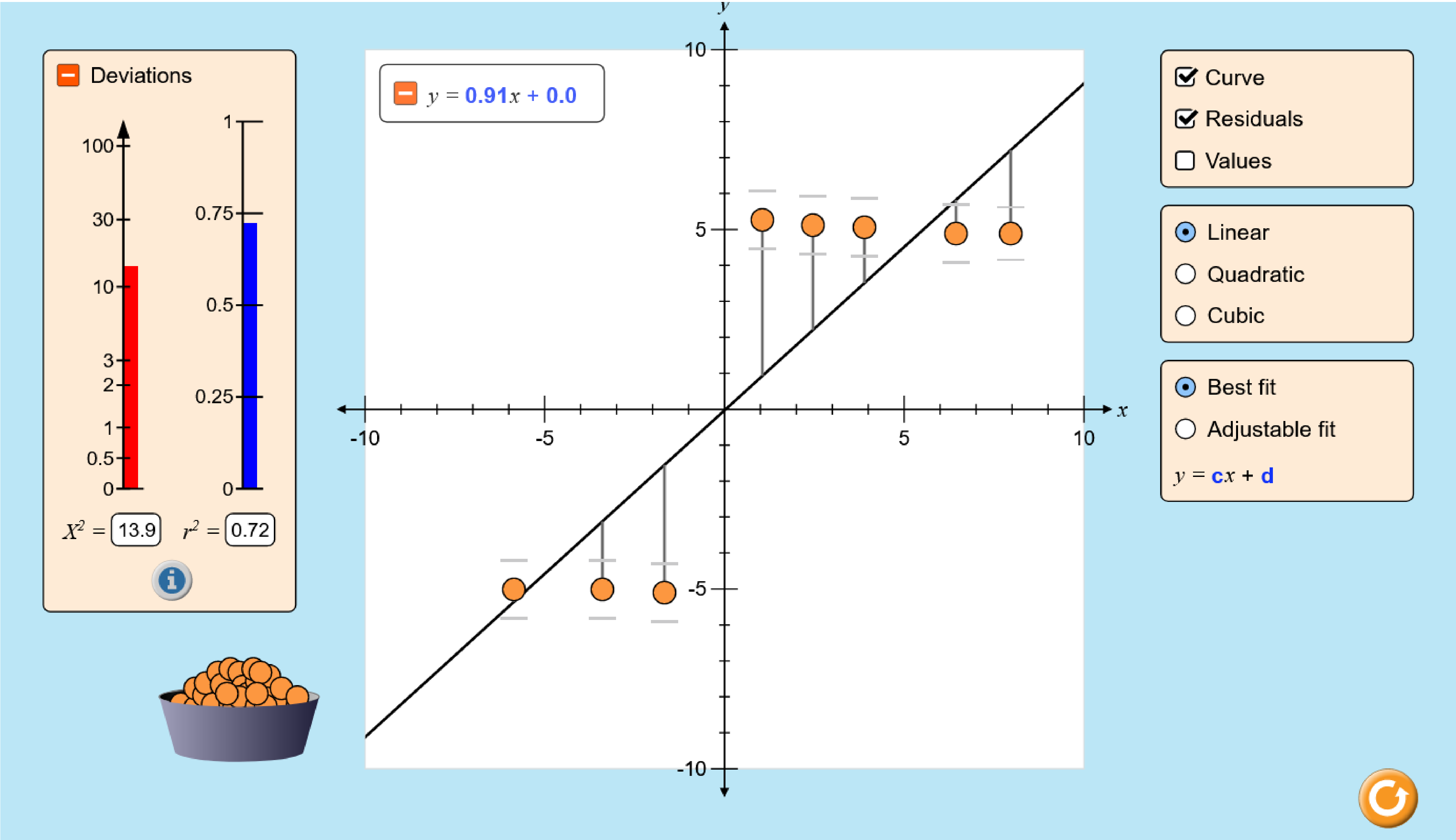
alpha : *float, default=1.0*

Regularization strength; must be a positive float. Regularization improves the conditioning of the problem and reduces the variance of the estimates. Larger values specify stronger regularization. Alpha corresponds to `1 / (2C)` in other linear models such as `LogisticRegression` or `LinearSVC`.

fit_intercept : *bool, default=True*

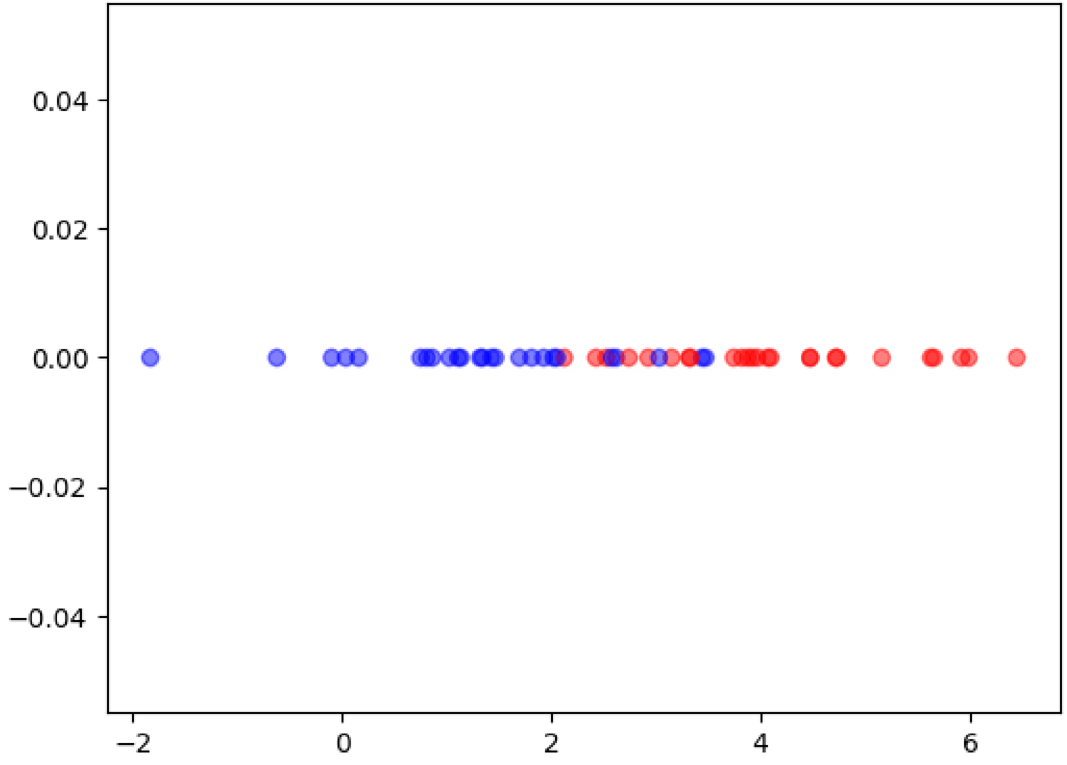
Whether to calculate the intercept for this model. If set to false, no intercept will be used in calculations (e.g. data is expected to be already centered).

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.RidgeClassifier.html#sklearn.linear_model.RidgeClassifier https://phet.colorado.edu/sims/html/curve-fitting/latest/curve-fitting_all.html

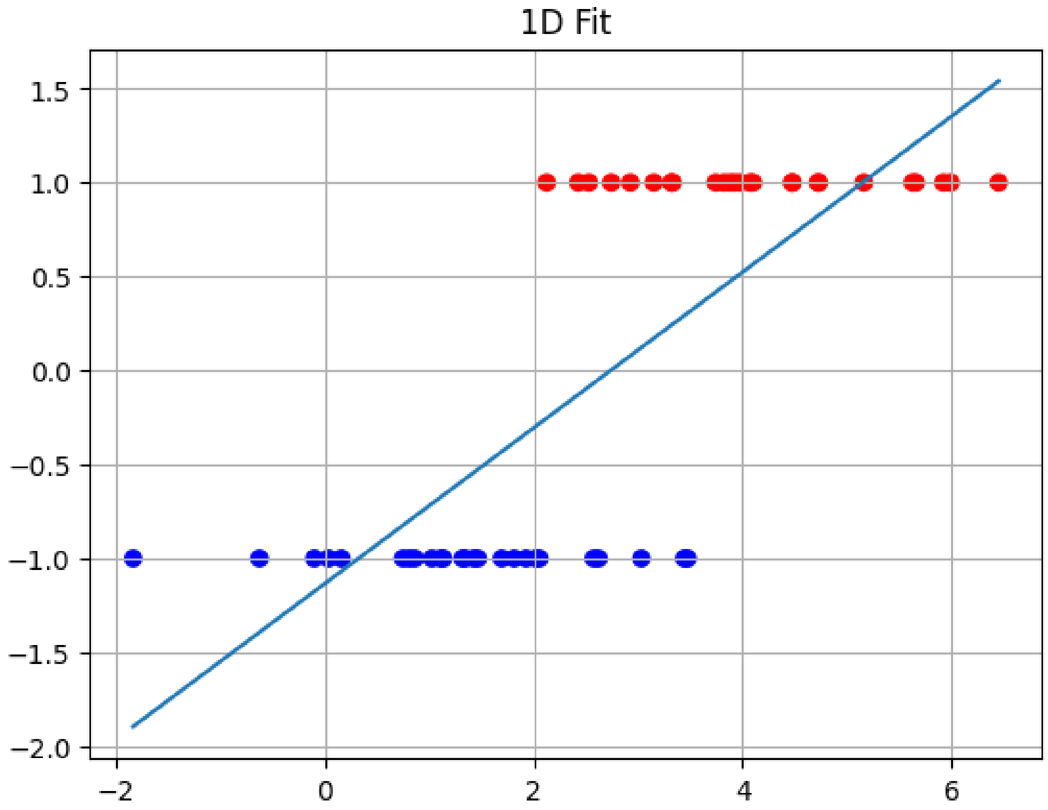


Single Feature Example

Plot of single feature data for two classes

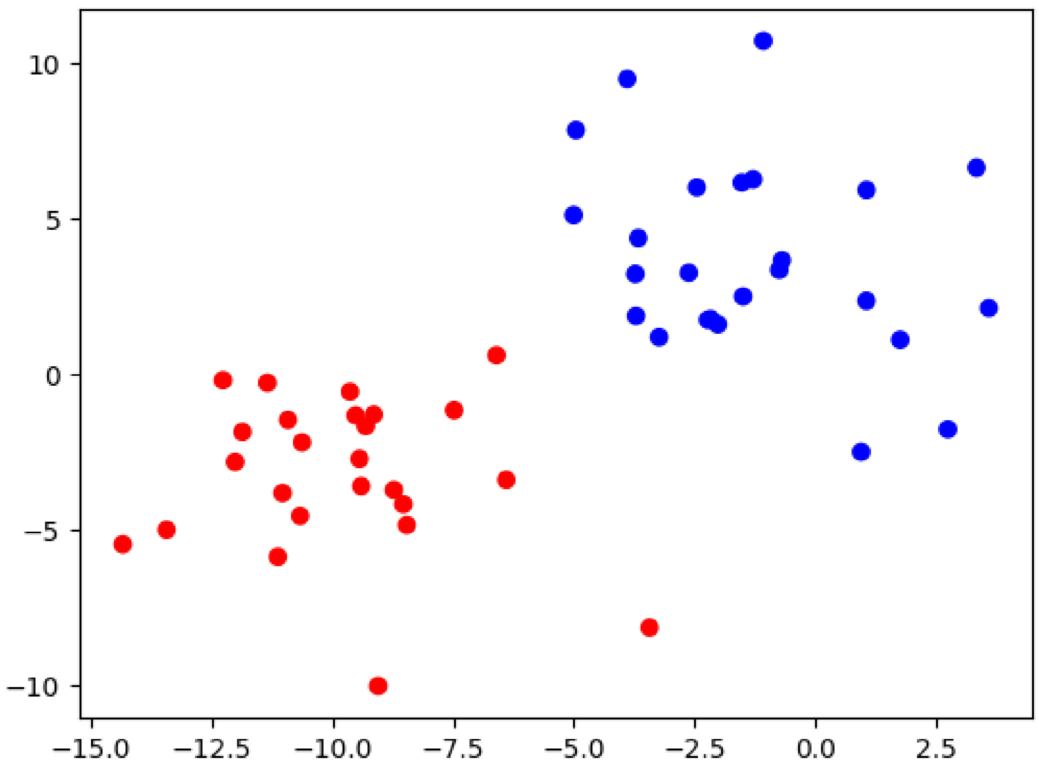


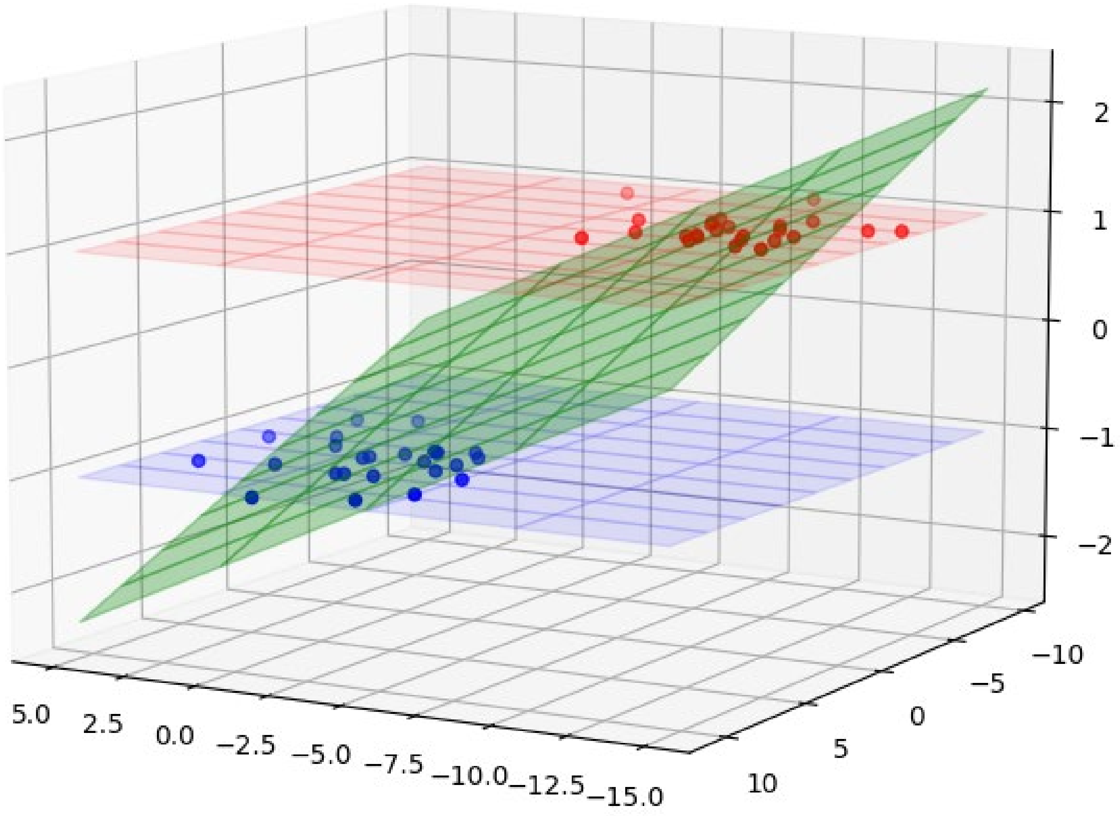
Set $y = 1$ for red, $y = -1$ for blue, fit a line



Decision boundary occurs at $y = 0$

Two Features



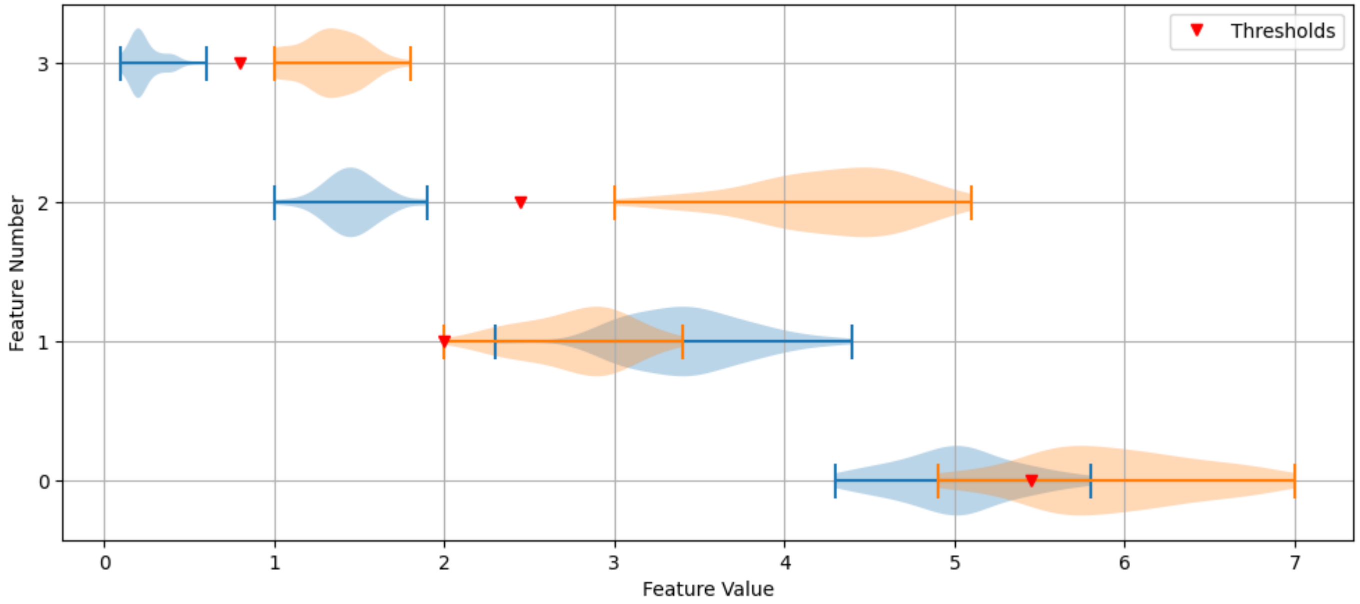


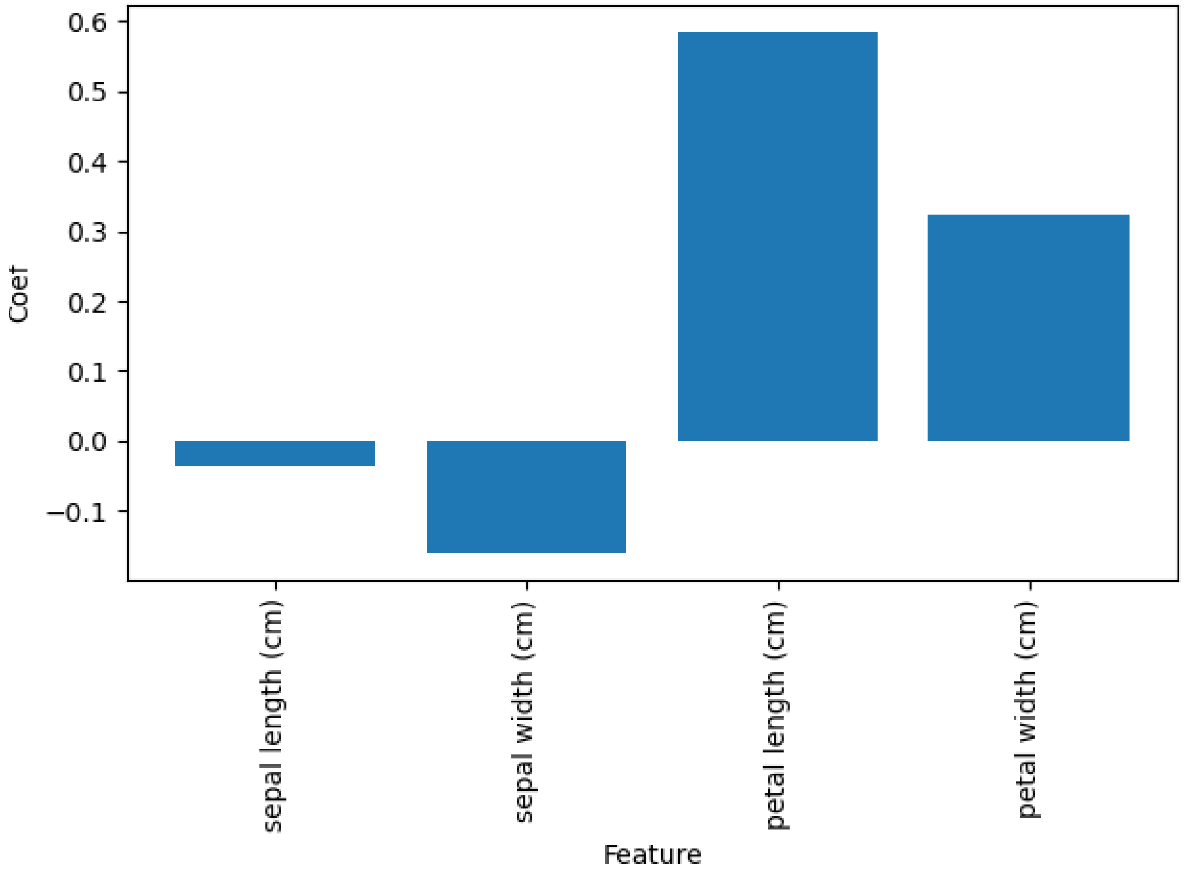
Feature Importance

If you have scaled the data, each feature's slope is like a rough estimate of feature importance

Be wary: it isn't always safe to interpret them this way. Issues like very non-linear data or highly-correlated features can break this.

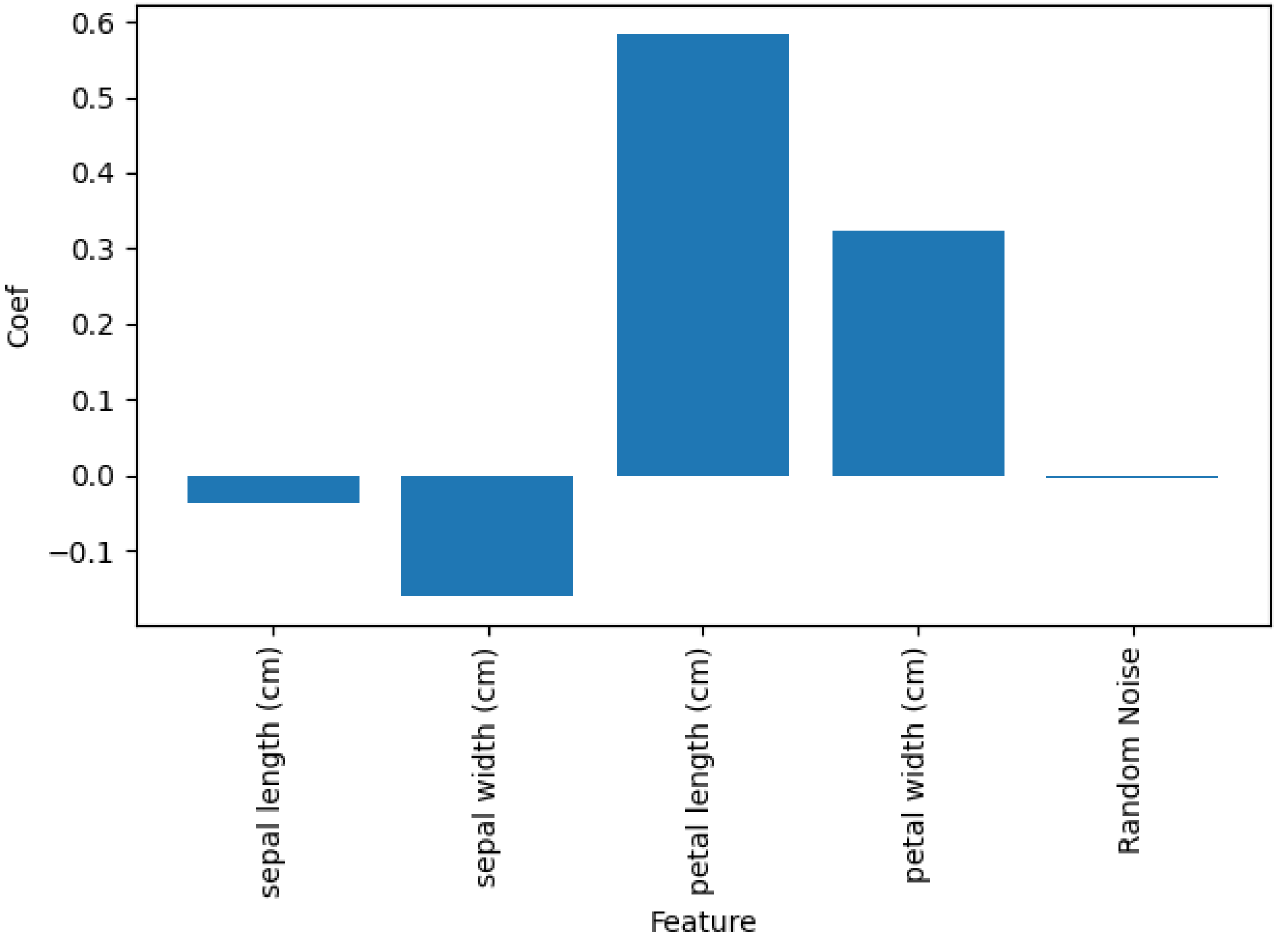
Example: Iris Data





Feature Importance

If you have scaled the data, each feature's slope is like a rough estimate of feature importance
Be wary: it isn't always safe to interpret them this way. Issues like very non-linear data or highly-correlated features can break this.



Example: Iris Data + Random Noise

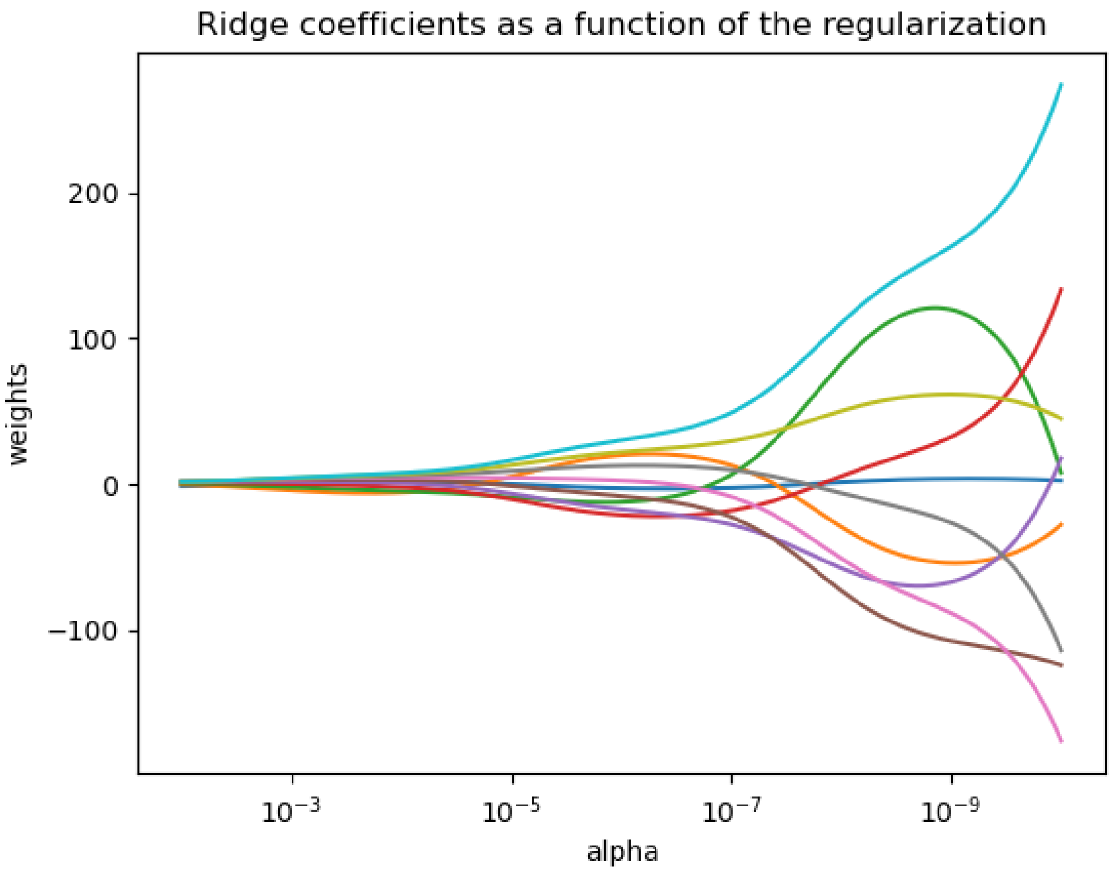
I made a new feature array with Iris (scaled and using only species 0 and 1) plus a fifth column of just random numbers. A linear fit shows that the coefficient is tiny on the random noise feature (i.e. this wasn't important for determining species)

NOTE: See Permutation Feature Importance for a systematic way to measure this for any model:
https://scikit-learn.org/stable/modules/permutation_importance.html

L2 Regularization

Fitting a line can sometimes cause numerical instabilities. When features are co-linear (or at least highly correlated) this almost always leads to huge problems. Solution is to penalize large weights.

https://scikit-learn.org/stable/auto_examples/linear_model/plot_ridge_path.html#sphx-glr-auto-examples-linear-model-plot-ridge-path-py



1.1.2.1. Regression

`Ridge` regression addresses some of the problems of Ordinary Least Squares by imposing a penalty on the size of the coefficients. The ridge coefficients minimize a penalized residual sum of squares:

$$\min_{w} || X w - y||_2^2 + \alpha ||w||_2^2$$

The complexity parameter $\alpha \geq 0$ controls the amount of shrinkage: the larger the value of α, the greater the amount of shrinkage and thus the coefficients become more robust to collinearity.

```
[62]  X.shape # iris data species 0 and 1 only
```

```
(100, 4)
```

```
[85]  X2 = np.c_[X, X[:,1]]  # duplicate column 1
      X2[:4]
```

Start with Iris, then duplicate column 1

```
array([[5.1, 3.5, 1.4, 0.2, 3.5],
       [4.9, 3. , 1.4, 0.2, 3. ],
       [4.7, 3.2, 1.3, 0.2, 3.2],
       [4.6, 3.1, 1.5, 0.2, 3.1]])
```

```
[89]  clf = linear_model.RidgeClassifier(alpha=0)
      clf.fit(X2,y)
      clf.coef_
```

Fit fails without L2 regularization ($\alpha = 0$)

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix (rcond=3.9912e-18): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
array([[-0.05697936,  3.66360497,  0.40626179,  0.57570033, -4.        ]])
```

```
[90]  clf = linear_model.RidgeClassifier(alpha=1)
      clf.fit(X2,y)
      clf.coef_
```

Works with L2 regularization ($\alpha = 1$)

```
array([[-0.04761258, -0.16538899,  0.46798125,  0.40226765, -0.16538899]])
```

L1 vs L2 Regularization

This method is called L2 regularization. Think of this as "sum of weights **squared**" to give us the 2. Advantages are stability and preventing overfitting.

We also have L1 regularization which is the sum of the absolute value of weights. The point is different: in L1 the goal is to drive some weights to zero and eliminate features!

Some classifiers do both L1 and L2:

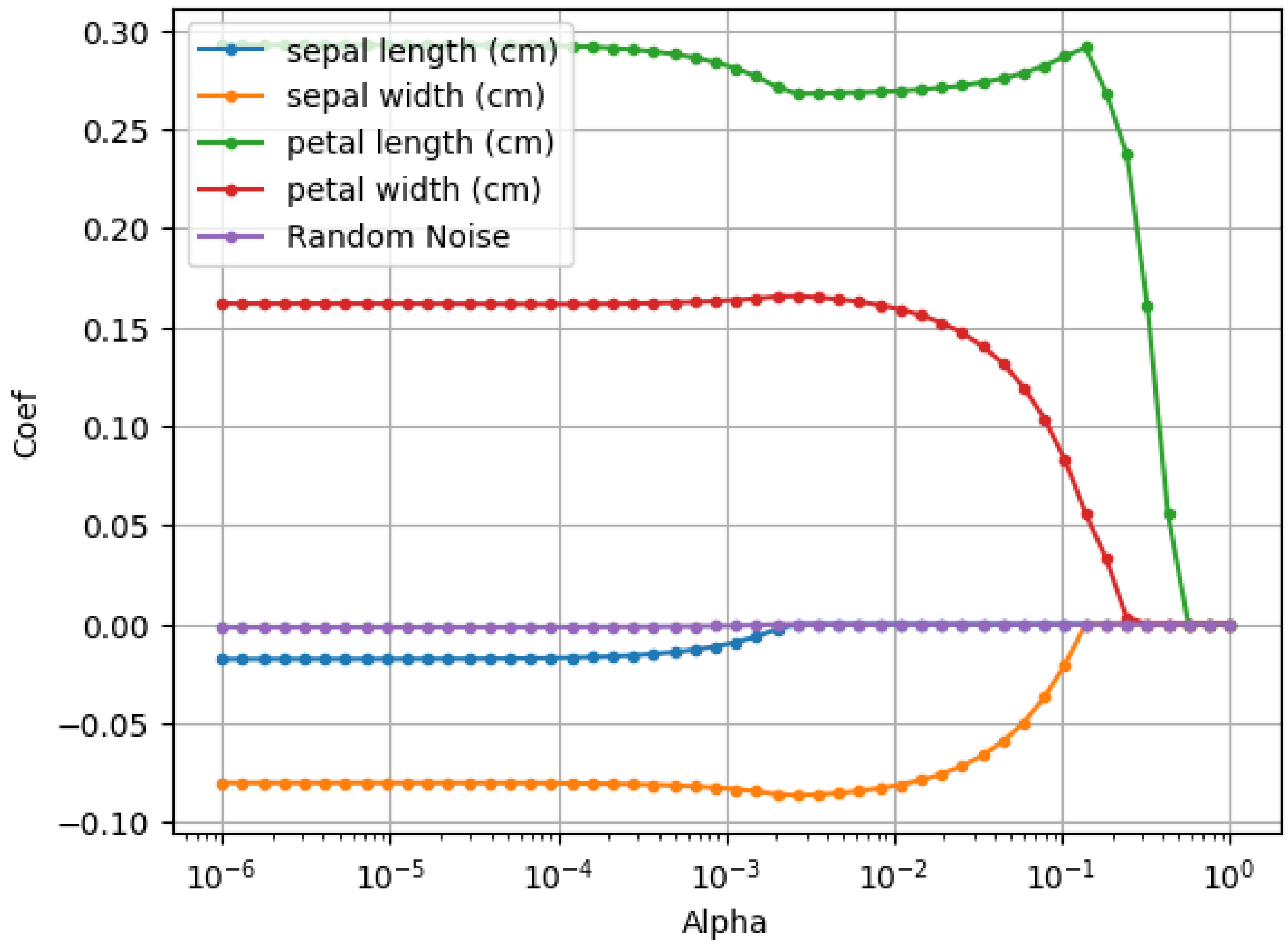
We currently provide four choices for the regularization term $r(w)$ via the `penalty` argument:

penalty	$r(w)$
`None`	0
ℓ_1	$\|w\|_1$
ℓ_2	$\frac{1}{2}\|w\|_2^2 = \frac{1}{2}w^T w$
`ElasticNet`	$\frac{1-\rho}{2}w^T w + \rho\|w\|_1$

For ElasticNet, ρ (which corresponds to the `l1_ratio` parameter) controls the strength of ℓ_1 regularization vs. ℓ_2 regularization. Elastic-Net is equivalent to ℓ_1 when $\rho = 1$ and equivalent to ℓ_2 when $\rho = 0$.

L1 Regularization

Iris Plus Random Noise

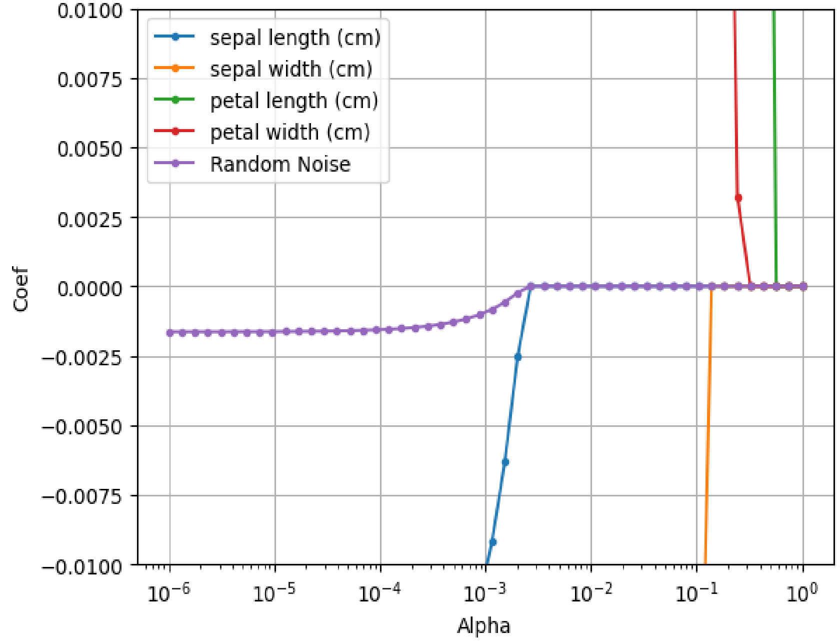



```python
ALPHAS = np.logspace(-6,0,50)
a = np.zeros( (len(ALPHAS),5))
for i in range(len(ALPHAS)):
  lasso = linear_model.Lasso(alpha=ALPHAS[i])
  lasso.fit(Xrs,y)  # scaled, last col is random noise
  #print(lasso.coef_)
  a[i] = lasso.coef_

for i,foo in enumerate(a.T):
  #print(foo)
  plt.semilogx(ALPHAS, foo,'.-', label=Xrs_features[i])

plt.legend(loc='upper left')
plt.grid()
plt.xlabel('Alpha')
plt.ylabel('Coef')
#plt.ylim(-0.002, 0.002)
plt.ylim(-0.01, 0.01)
plt.show()
```

Zoomed

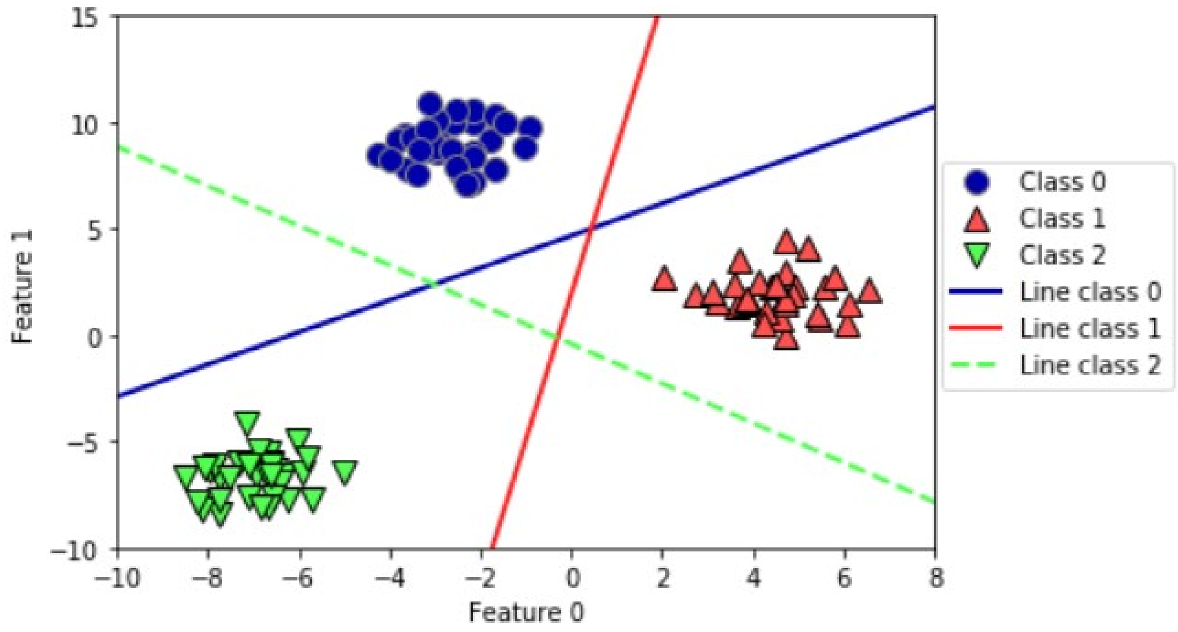


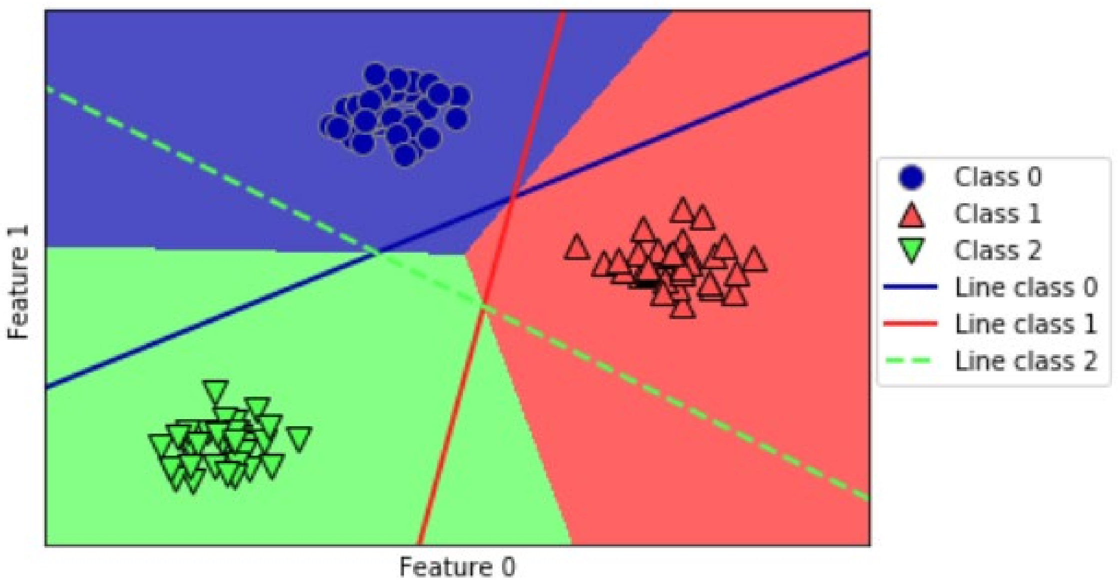
- Fit lasso with L1 to Iris+Random Noise data for different α
- Nothing happens for $\alpha < 0.001$
- Increasing α starts setting coefficients to 0 starting with random noise and sepal length

Multiple Targets

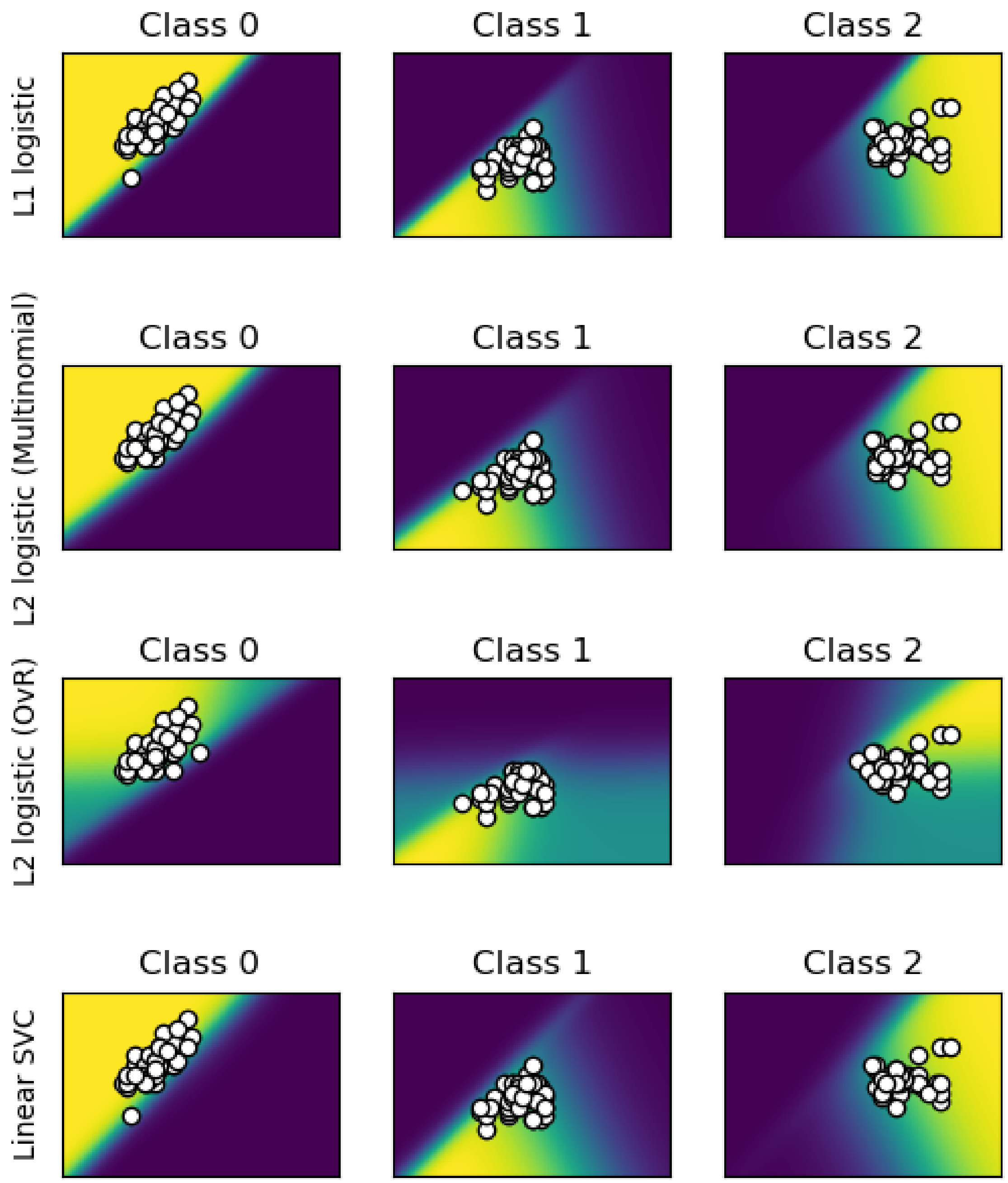
How do we handle multiple targets?

- One-vs-All (OvA) also called One-vs-Rest (OvR) is the common approach.
- Train a different classifier for each target that tries to separate one target from all of the others
- To make a prediction all classifiers score and the most likely one wins



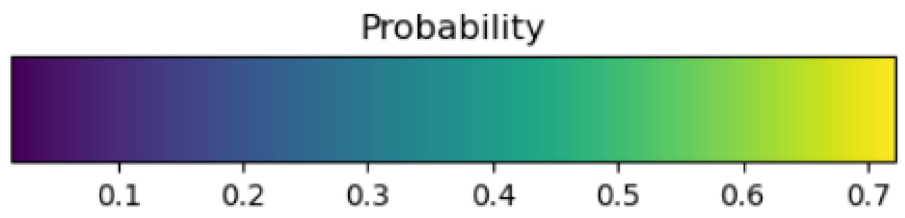


https://github.com/amueller/introduction_to_ml_with_python/blob/master/02-supervised-learning.ipynb https://scikit-learn.org/stable/auto_examples/classification/plot_classification_probability.html#sphx-glr-auto-examples-classification-plot-classification-probability-py



"Plot the classification probability for different classifiers. We use a 3 class dataset, and we classify it with a Support Vector classifier, L1 and L2 penalized logistic regression with either a One-Vs-Rest or multinomial setting, and Gaussian process classification. "

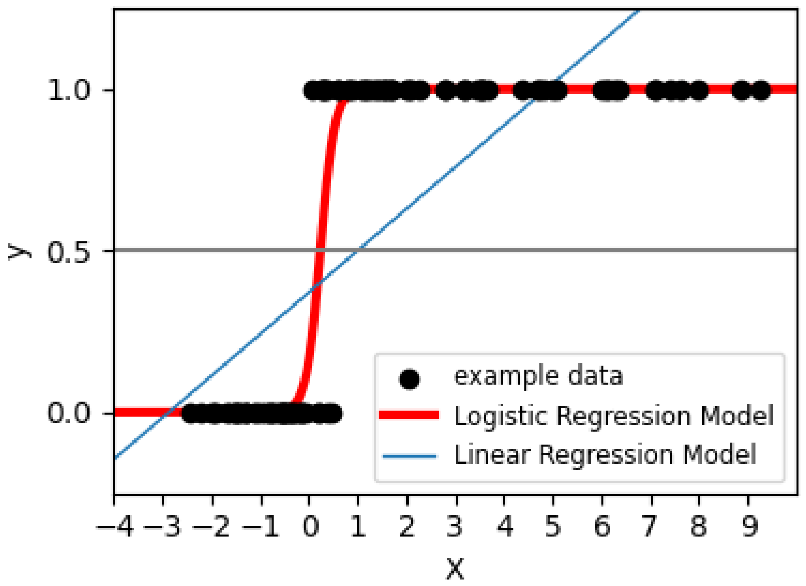
Classification works by fitting models to $y_i = 1$ for target i and $y = -1$ for other targets. For Iris we fit 3 separate models. To classify we simply calculate all 3 y values and choose the biggest (no voting—winner takes all)



1.1.11. Logistic regression ¶

The logistic regression is implemented in `LogisticRegression`. Despite its name, it is implemented as a linear model for classification rather than regression in terms of the scikit-learn/ML nomenclature. The logistic regression is also known in the literature as logit regression, maximum-entropy classification (MaxEnt) or the log-linear classifier. In this model, the probabilities describing the possible outcomes of a single trial are modeled using a logistic function.

This implementation can fit binary, One-vs-Rest, or multinomial logistic regression with optional ℓ_1, ℓ_2 or Elastic-Net regularization.



https://scikit-learn.org/stable/auto_examples/linear_model/plot_logistic.html#sphx-glr-auto-examples-linear-model-plot-logistic-py

1.1.11.1. Binary Case

For notational ease, we assume that the target y_i takes values in the set $\{0, 1\}$ for data point i. Once fitted, the `predict_proba` method of `LogisticRegression` predicts the probability of the positive class $P(y_i = 1|X_i)$ as

$$\hat{p}(X_i) = \operatorname{expit}(X_i w + w_0) = \frac{1}{1 + \exp(-X_i w - w_0)}.$$

As an optimization problem, binary class logistic regression with regularization term $r(w)$ minimizes the following cost function:

$$\min_w C \sum_{i=1}^{n} \left(-y_i \log(\hat{p}(X_i)) - (1 - y_i) \log(1 - \hat{p}(X_i))\right) + r(w).$$

We currently provide four choices for the regularization term $r(w)$ via the `penalty` argument:

penalty	$r(w)$
`None`	0
ℓ_1	$\|w\|_1$
ℓ_2	$\frac{1}{2}\|w\|_2^2 = \frac{1}{2}w^T w$
`ElasticNet`	$\frac{1-\rho}{2}w^T w + \rho\|w\|_1$

For ElasticNet, ρ (which corresponds to the `l1_ratio` parameter) controls the strength of ℓ_1 regularization vs. ℓ_2 regularization. Elastic-Net is equivalent to ℓ_1 when $\rho = 1$ and equivalent to ℓ_2 when $\rho = 0$.

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html#sklearn.linear_model.LogisticRegression

sklearn.linear_model.LogisticRegression ¶

class `sklearn.linear_model.`**`LogisticRegression`**(*penalty='l2', *, dual=False, tol=0.0001, C=1.0, fit_intercept=True, intercept_scaling=1, class_weight=None, random_state=None, solver='lbfgs', max_iter=100, multi_class='auto', verbose=0, warm_start=False, n_jobs=None, l1_ratio=None*) [source]

Logistic Regression (aka logit, MaxEnt) classifier.

In the multiclass case, the training algorithm uses the one-vs-rest (OvR) scheme if the 'multi_class' option is set to 'ovr', and uses the cross-entropy loss if the 'multi_class' option is set to 'multinomial'. (Currently the 'multinomial' option is supported only by the 'lbfgs', 'sag', 'saga' and 'newton-cg' solvers.)

This class implements regularized logistic regression using the 'liblinear' library, 'newton-cg', 'sag', 'saga' and 'lbfgs' solvers. **Note that regularization is applied by default**. It can handle both dense and sparse input. Use C-ordered arrays or CSR matrices containing 64-bit floats for optimal performance; any other input format will be converted (and copied).

The 'newton-cg', 'sag', and 'lbfgs' solvers support only L2 regularization with primal formulation, or no regularization. The 'liblinear' solver supports both L1 and L2 regularization, with a dual formulation only for the L2 penalty. The Elastic-Net regularization is only supported by the 'saga' solver.

Read more in the User Guide.

Parameters:

penalty : *{'l1', 'l2', 'elasticnet', None}, default='l2'*

Specify the norm of the penalty:

- `None`: no penalty is added;
- `'l2'`: add a L2 penalty term and it is the default choice;
- `'l1'`: add a L1 penalty term;
- `'elasticnet'`: both L1 and L2 penalty terms are added.

multi_class : *{'auto', 'ovr', 'multinomial'}, default='auto'*

If the option chosen is 'ovr', then a binary problem is fit for each label. For 'multinomial' the loss minimised is the multinomial loss fit across the entire probability distribution, *even when the data is binary*. 'multinomial' is unavailable when solver='liblinear'. 'auto' selects 'ovr' if the data is binary, or if solver='liblinear', and otherwise selects 'multinomial'.

New in version 0.18: Stochastic Average Gradient descent solver for 'multinomial' case.

Changed in version 0.22: Default changed from 'ovr' to 'auto' in 0.22.

Classifier	Regularization	Parameters	Notes
RidgeClassifier	L2	Alpha = 1.0	
RidgeClassifierCV	L2	Alpha=(0.1, 1.0, 10.0)	
Lasso	L1	Alpha=1.0	
LassoCV	L1	Alpha found automatically	Automatic search instead of grid
ElasticNet	Both	Alpha=1.0, l1_ratio=0.5	L1_ratio=1 is pure L1 L1_ratio=0 is pure L2
ElasticNetCV	Both	L1_ratio =0.5, alpha found automatically	List of good L1_ratios to try [.1, .5, .7, .9, .95, .99, 1] (i.e. don’t get too close to 0)
LogisticRegression	Both	L1_ratio C=1.0	Alpha=1/2C. No reg = Large C
LogisticRegressionCV	Both	Cs defaults to 10 in range [1e-4, 1e4] L1_ratios must specify	

Regression Summary

Set target values $y = [+1, -1]$ and fit a line!

New Concepts:

- For >2 targets use One-vs-All (OvA) scheme
  - *Sometimes this is called one-vs-rest*
- Regularization:
  - L2 penalize large weights, vital when features are co-linear
  - L1 remove worthless features

Classifiers to try:

- Simple: RidgeClassifierCV
- To remove features: LassoCV
- Fancy: ElasticNetCV or LogisticRegressionCV

Search Models

Stochastic Gradient Descent

https://scikit-learn.org/stable/modules/sgd.html#sgd

1.5. Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) is a simple yet very efficient approach to fitting linear classifiers and regressors under convex loss functions such as (linear) Support Vector Machines and Logistic Regression. Even though SGD has been around in the machine learning community for a long time, it has received a considerable amount of attention just recently in the context of large-scale learning.

SGD has been successfully applied to large-scale and sparse machine learning problems often encountered in text classification and natural language processing. Given that the data is sparse, the classifiers in this module easily scale to problems with more than 10^5 training examples and more than 10^5 features.

Strictly speaking, SGD is merely an optimization technique and does not correspond to a specific family of machine learning models. It is only a *way* to train a model. Often, an instance of `SGDClassifier` or `SGDRegressor` will have an equivalent estimator in the scikit-learn API, potentially using a different optimization technique. For example, using `SGDClassifier(loss='log_loss')` results in logistic regression, i.e. a model equivalent to `LogisticRegression` which is fitted via SGD instead of being fitted by one of the other solvers in `LogisticRegression`. Similarly, `SGDRegressor(loss='squared_error', penalty='l2')` and `Ridge` solve the same optimization problem, via different means.

The advantages of Stochastic Gradient Descent are:

- Efficiency.
- Ease of implementation (lots of opportunities for code tuning).

The disadvantages of Stochastic Gradient Descent include:

- SGD requires a number of hyperparameters such as the regularization parameter and the number of iterations.
- SGD is sensitive to feature scaling.

1.5.8. Mathematical formulation

We describe here the mathematical details of the SGD procedure. A good overview with convergence rates can be found in [12].

Given a set of training examples $(x_1, y_1), \ldots, (x_n, y_n)$ where $x_i \in \mathbf{R}^m$ and $y_i \in \mathcal{R}$ ($y_i \in -1, 1$ for classification), our goal is to learn a linear scoring function $f(x) = w^T x + b$ with model parameters $w \in \mathbf{R}^m$ and intercept $b \in \mathbf{R}$. In order to make predictions for binary classification, we simply look at the sign of $f(x)$. To find the model parameters, we minimize the regularized training error given by

$$E(w, b) = \frac{1}{n} \sum_{i=1}^{n} L(y_i, f(x_i)) + \alpha R(w)$$

where L is a loss function that measures model (mis)fit and R is a regularization term (aka penalty) that penalizes model complexity; $\alpha > 0$ is a non-negative hyperparameter that controls the regularization strength.

Different choices for L entail different classifiers or regressors:

- Hinge (soft-margin): equivalent to Support Vector Classification. $L(y_i, f(x_i)) = \max(0, 1 - y_i f(x_i))$.
- Perceptron: $L(y_i, f(x_i)) = \max(0, -y_i f(x_i))$.
- Modified Huber: $L(y_i, f(x_i)) = \max(0, 1 - y_i f(x_i))^2$ if $y_i f(x_i) > 1$, and $L(y_i, f(x_i)) = -4 y_i f(x_i)$ otherwise.
- Log Loss: equivalent to Logistic Regression. $L(y_i, f(x_i)) = \log(1 + \exp(-y_i f(x_i)))$.
- Squared Error: Linear regression (Ridge or Lasso depending on R). $L(y_i, f(x_i)) = \frac{1}{2}(y_i - f(x_i))^2$.
- Huber: less sensitive to outliers than least-squares. It is equivalent to least squares when $|y_i - f(x_i)| \leq \varepsilon$, and $L(y_i, f(x_i)) = \varepsilon |y_i - f(x_i)| - \frac{1}{2}\varepsilon^2$ otherwise.
- Epsilon-Insensitive: (soft-margin) equivalent to Support Vector Regression. $L(y_i, f(x_i)) = \max(0, |y_i - f(x_i)| - \varepsilon)$.

Important History: PERCEPTRON

Define a loss function where we penalize missed points. This leads to a simple and efficient algorithm called the perceptron

Popular choices for the regularization term R (the `penalty` parameter) include:

- L2 norm: $R(w) := \frac{1}{2} \sum_{j=1}^{m} w_j^2 = ||w||_2^2$,
- L1 norm: $R(w) := \sum_{j=1}^{m} |w_j|$, which leads to sparse solutions.
- Elastic Net: $R(w) := \frac{\rho}{2} \sum_{j=1}^{n} w_j^2 + (1 - \rho) \sum_{j=1}^{m} |w_j|$, a convex combination of L2 and L1, where ρ is given by `1 - l1_ratio`.

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html#sklearn.linear_model.SGDClassifier

sklearn.linear_model.SGDClassifier

class `sklearn.linear_model.SGDClassifier`(*loss='hinge', *, penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=1000, tol=0.001, shuffle=True, verbose=0, epsilon=0.1, n_jobs=None, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5, early_stopping=False, validation_fraction=0.1, n_iter_no_change=5, class_weight=None, warm_start=False, average=False*) [source]

Parameters:

loss : *{'hinge', 'log_loss', 'log', 'modified_huber', 'squared_hinge', 'perceptron', 'squared_error', 'huber', 'epsilon_insensitive', 'squared_epsilon_insensitive'}, default='hinge'*

The loss function to be used.

- 'hinge' gives a linear SVM.
- 'log_loss' gives logistic regression, a probabilistic classifier.
- **'modified_huber' is another smooth loss that brings tolerance to**

  outliers as well as probability estimates.

- 'squared_hinge' is like hinge but is quadratically penalized.
- 'perceptron' is the linear loss used by the perceptron algorithm.
- The other losses, 'squared_error', 'huber', 'epsilon_insensitive' and 'squared_epsilon_insensitive' are designed for regression but can be useful in classification as well; see `SGDRegressor` for a description.

More details about the losses formulas can be found in the User Guide.

Deprecated since version 1.1: The loss 'log' was deprecated in v1.1 and will be removed in version 1.3. Use `loss='log_loss'` which is equivalent.

penalty : *{'l2', 'l1', 'elasticnet', None}, default='l2'*

The penalty (aka regularization term) to be used. Defaults to 'l2' which is the standard regularizer for linear SVM models. 'l1' and 'elasticnet' might bring sparsity to the model (feature selection) not achievable with 'l2'. No penalty is added when set to `None`.

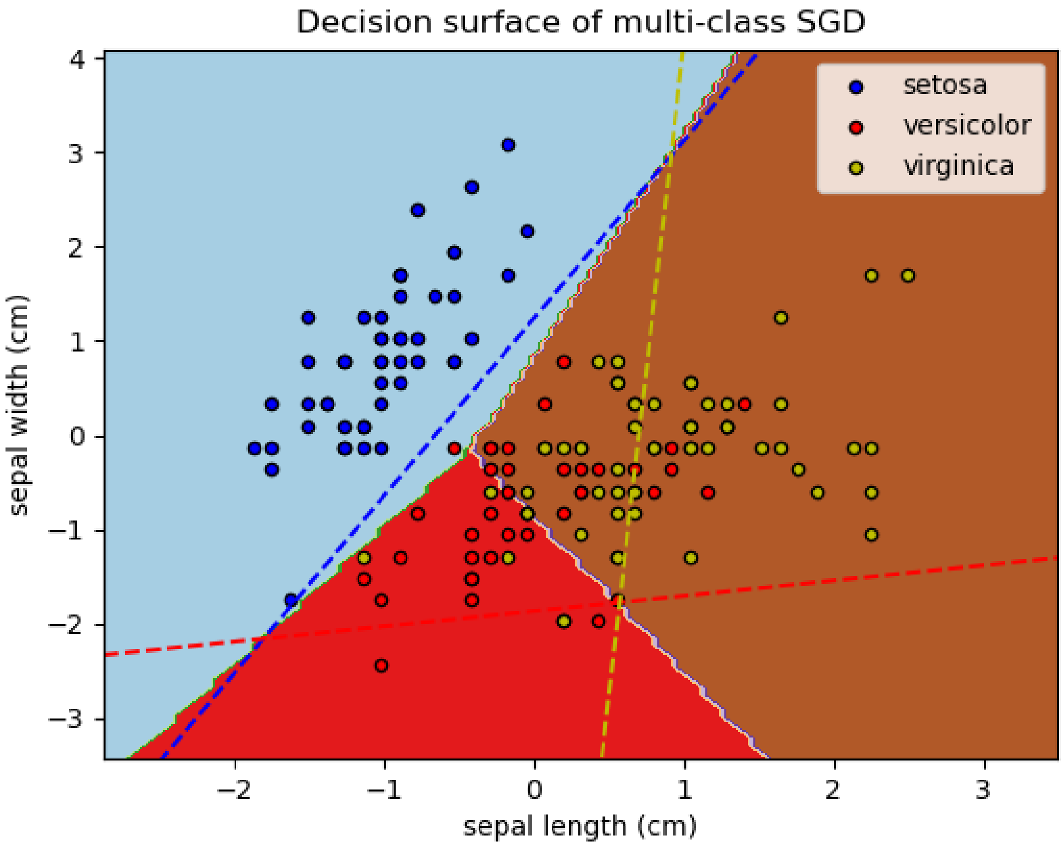
alpha : *float, default=0.0001*

Constant that multiplies the regularization term. The higher the value, the stronger the regularization. Also used to compute the learning rate when `learning_rate` is set to 'optimal'. Values must be in the range `[0.0, inf)`.

l1_ratio : *float, default=0.15*

The Elastic Net mixing parameter, with 0 <= l1_ratio <= 1. l1_ratio=0 corresponds to L2 penalty, l1_ratio=1 to L1. Only used if `penalty` is 'elasticnet'. Values must be in the range `[0.0, 1.0]`.

https://scikit-learn.org/stable/auto_examples/linear_model/plot_sgd_iris.html



`SGDClassifier` supports multi-class classification by combining multiple binary classifiers in a "one versus all" (OVA) scheme. For each of the K classes, a binary classifier is learned that discriminates between that and all other $K-1$ classes. At testing time, we compute the confidence score (i.e. the signed distances to the hyperplane) for each classifier and choose the class with the highest confidence. The Figure below illustrates the OVA approach on the iris dataset. The dashed lines represent the three OVA classifiers; the background colors show the decision surface induced by the three classifiers.

Search Models

Define generic error function and minimize it!

New Concepts

- Gradient Descent: greedy algorithm minimizing a function using Cal3

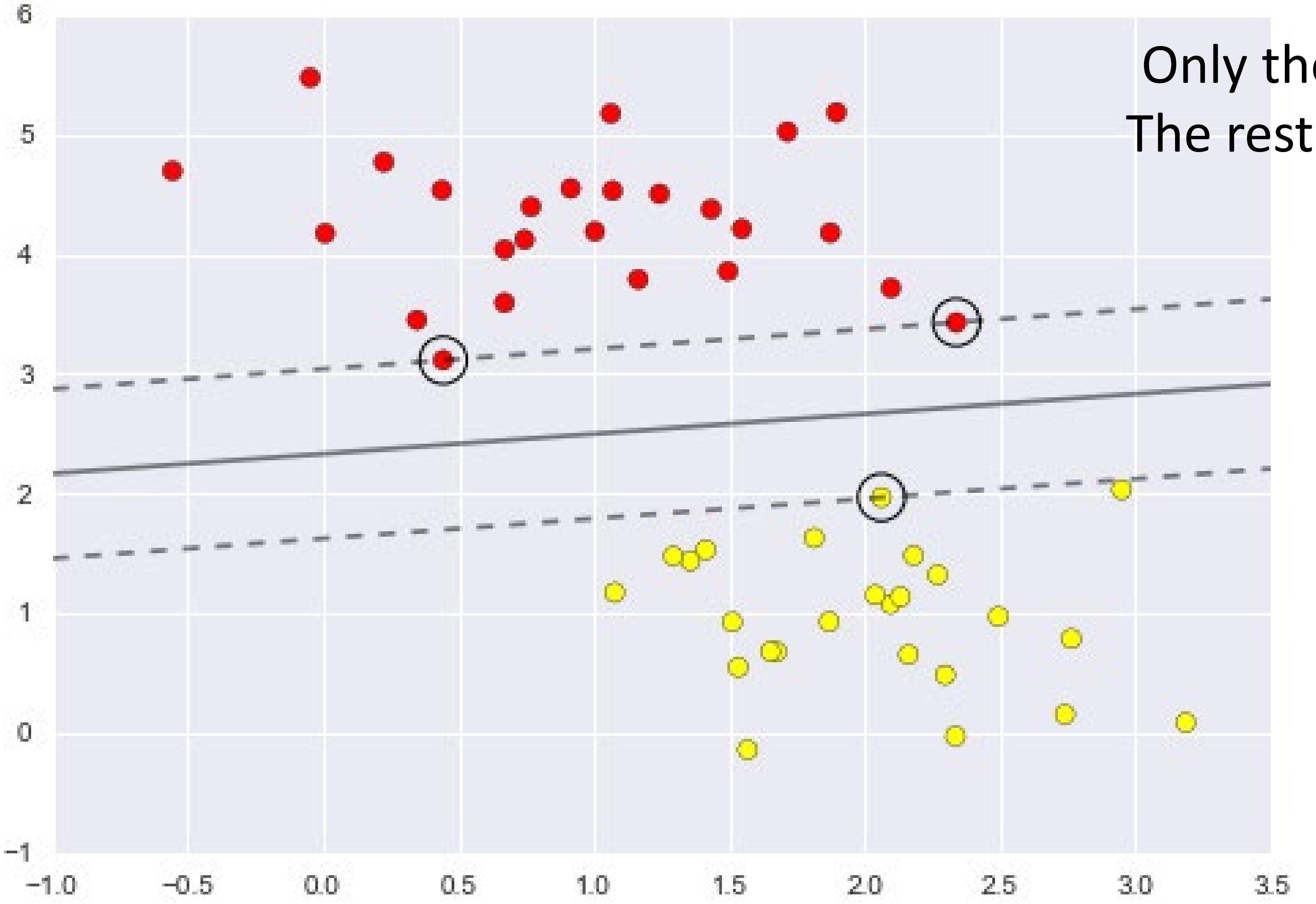
Classifiers

- SGDClassifier: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html#sklearn.linear_model.SGDClassifier

Margin Models

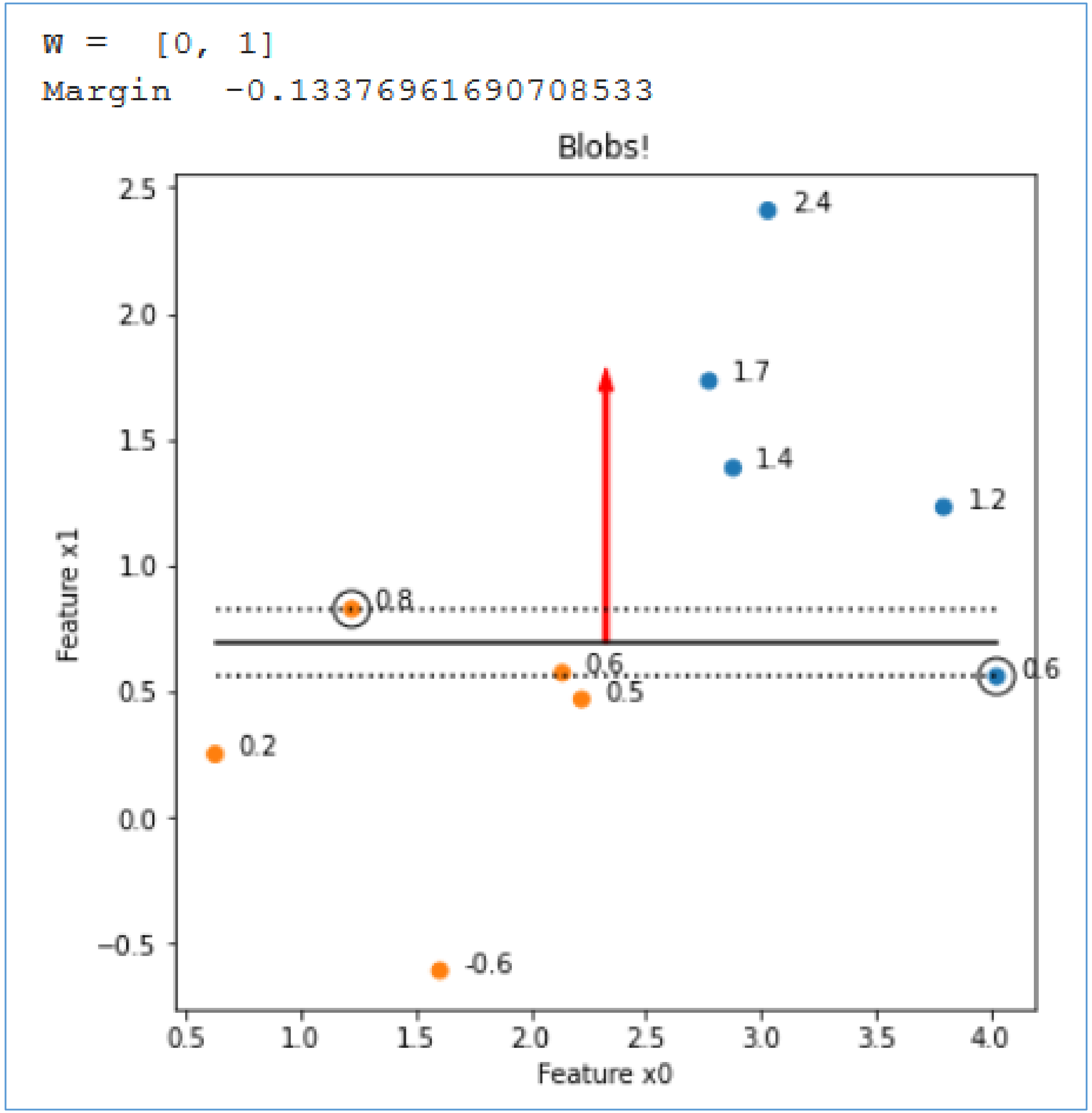
The Support Vector Machine

https://jakevdp.github.io/PythonDataScienceHandbook/05.07-support-vector-machines.html



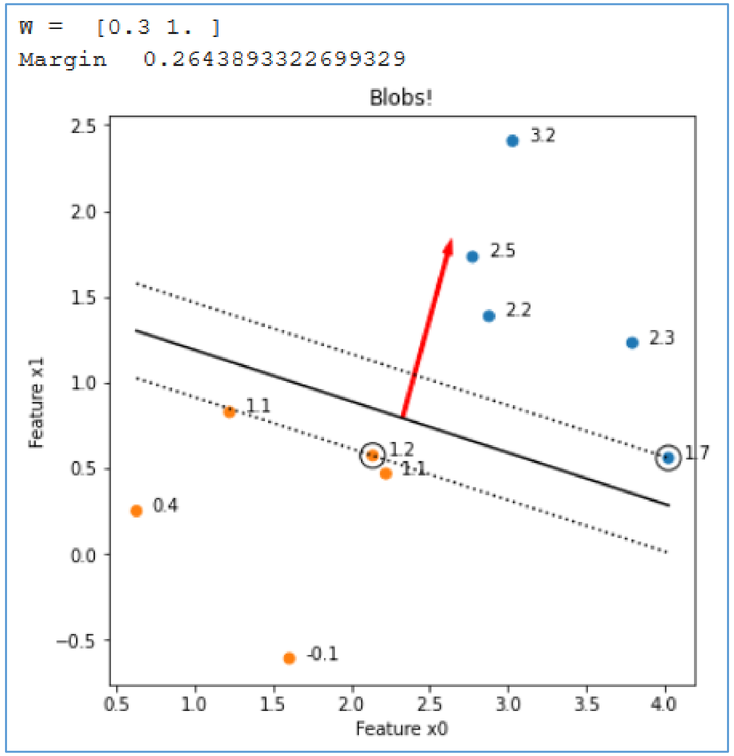
Only the support vectors matter!
The rest of the points are ignored!

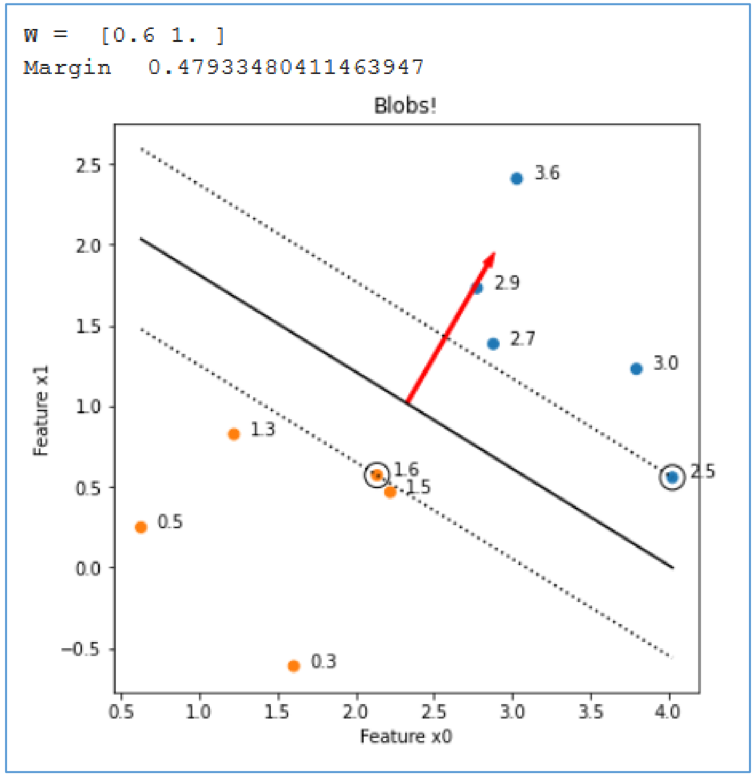
SVM Algorithm

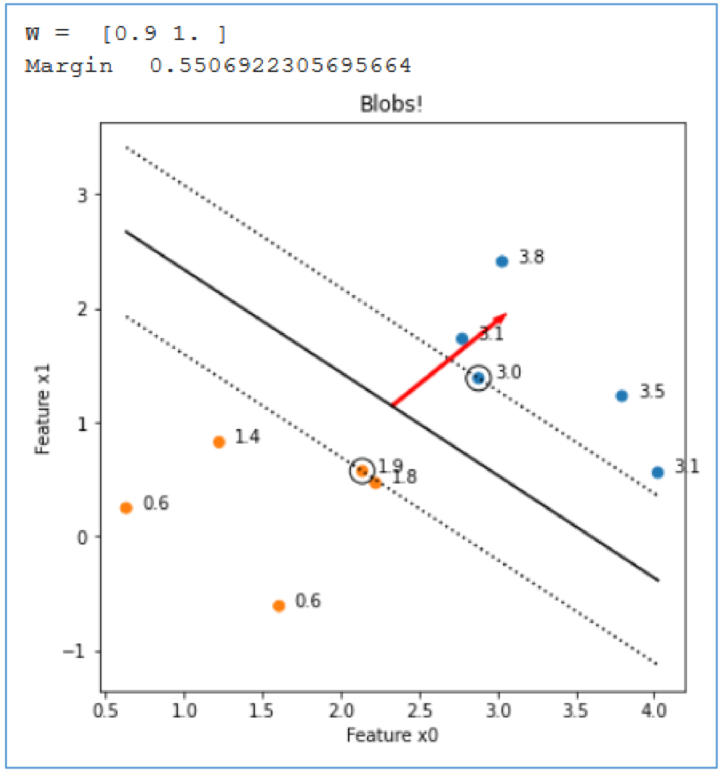


0) Start with random w
1) Calculate $X \cdot w$ as measure of distance from origin to each point parallel to w
2) Find support vectors (min and max distances for each target)
3) Margin $m = (d_{min} - d_{max})/2$ and $b = d_{max} + m$
4) Iteratively adjust w to maximize margin and repeat

SVM Algorithm

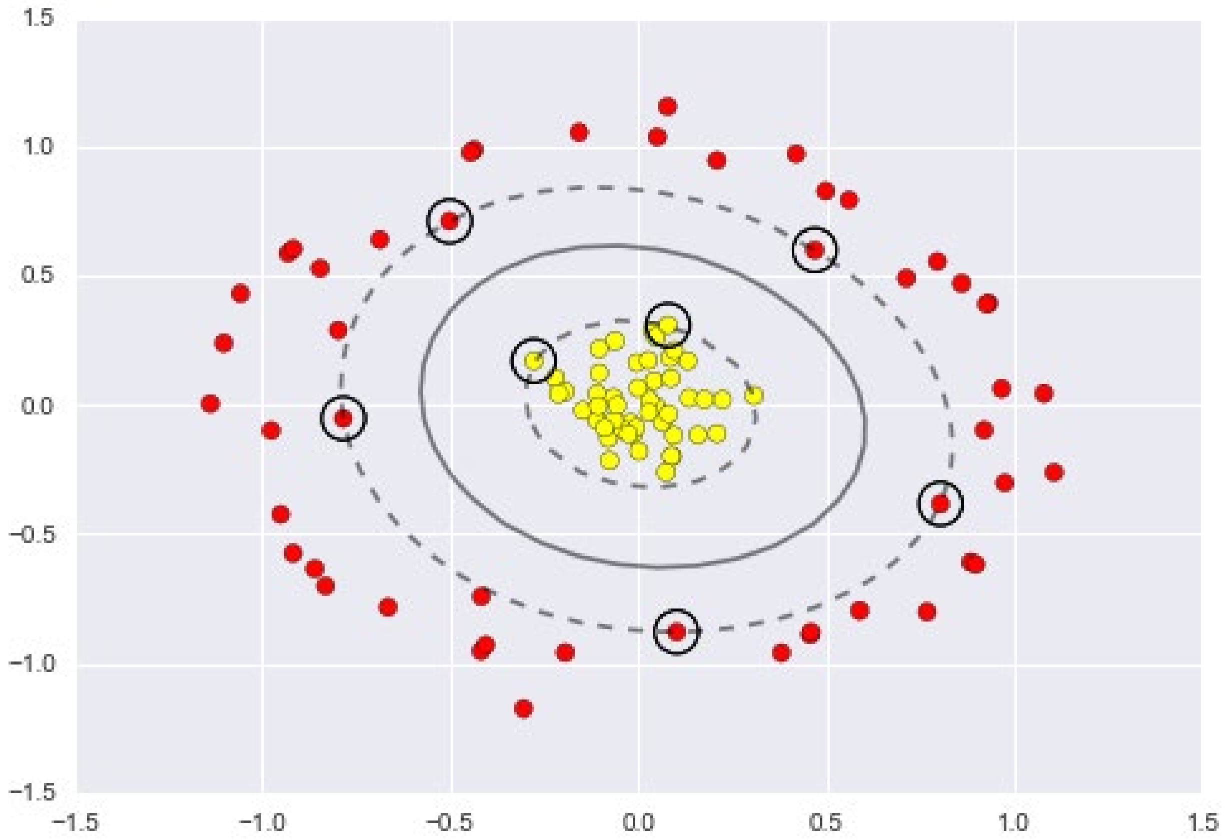


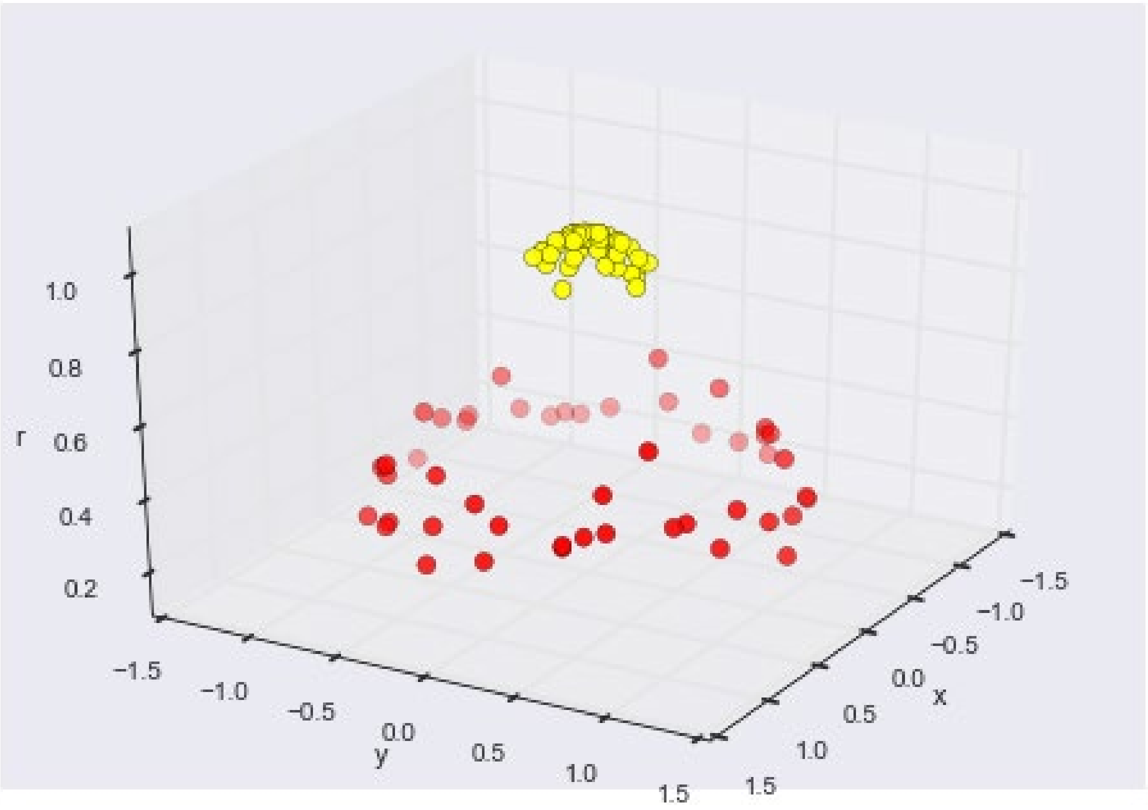




https://jakevdp.github.io/PythonDataScienceHandbook/05.07-support-vector-machines.html

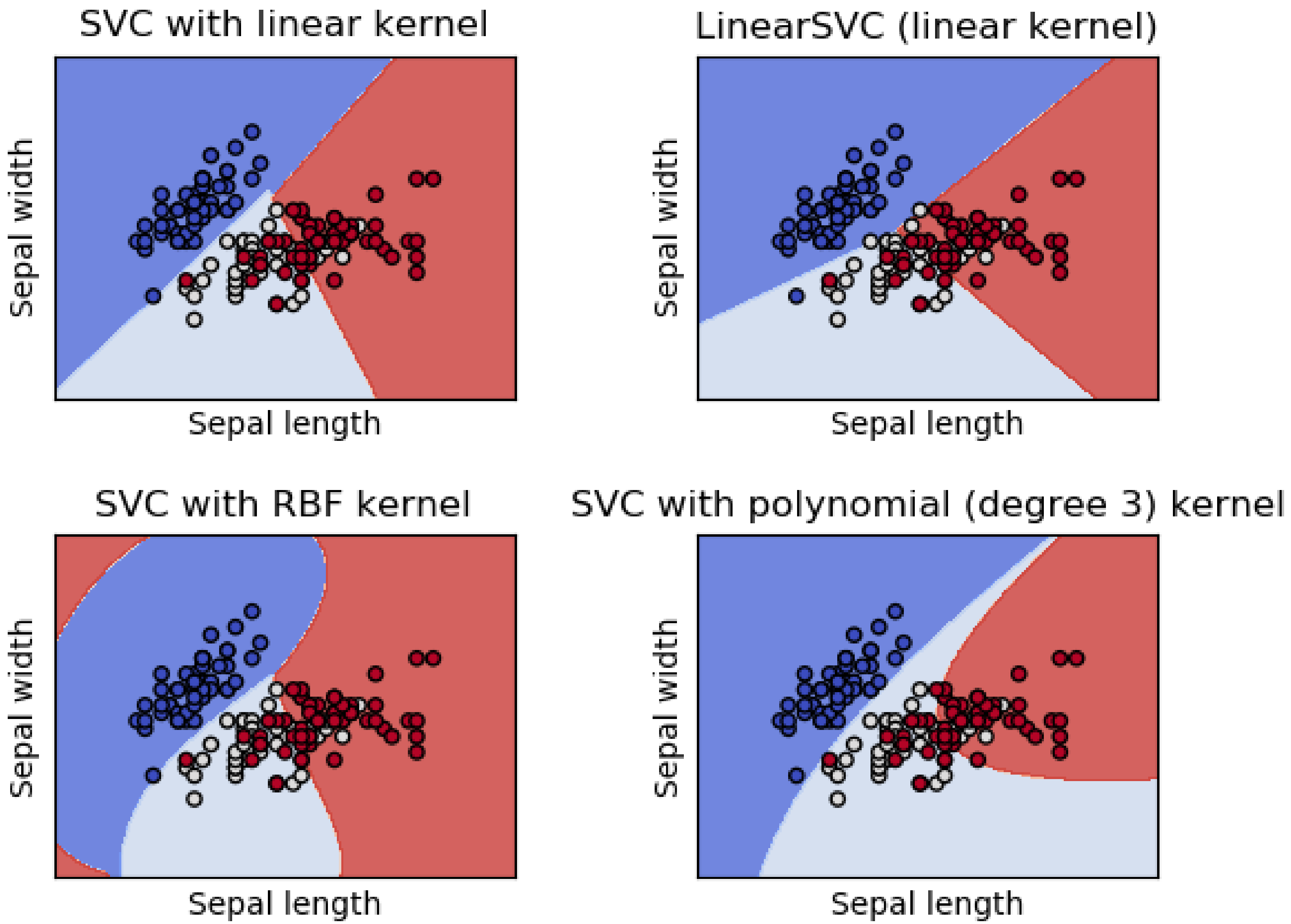
Non-linear problems? Project into higher space and do linear fit there!





RBF kernel

https://scikit-learn.org/stable/auto_examples/svm/plot_iris_svc.html#sphx-glr-auto-examples-svm-plot-iris-svc-py



https://scikit-learn.org/stable/modules/svm.html

1.4.1.1. Multi-class classification

`SVC` and `NuSVC` implement the "one-versus-one" approach for multi-class classification. In total, `n_classes * (n_classes - 1) / 2` classifiers are constructed and each one trains data from two classes. To provide a consistent interface with other classifiers, the `decision_function_shape` option allows to monotonically transform the results of the "one-versus-one" classifiers to a "one-vs-rest" decision function of shape `(n_samples, n_classes)`.

```
>>> X = [[0], [1], [2], [3]]
>>> Y = [0, 1, 2, 3]
>>> clf = svm.SVC(decision_function_shape='ovo')
>>> clf.fit(X, Y)
SVC(decision_function_shape='ovo')
>>> dec = clf.decision_function([[1]])
>>> dec.shape[1] # 4 classes: 4*3/2 = 6
6
>>> clf.decision_function_shape = "ovr"
>>> dec = clf.decision_function([[1]])
>>> dec.shape[1] # 4 classes
4
```

On the other hand, `LinearSVC` implements "one-vs-the-rest" multi-class strategy, thus training `n_classes` models.

Margin Models

Only focus on points at boundary, ignore everything else!

New Concepts

- Non-linear Kernels: like in LogisticRegression, we can add a non-linear function to the features (or even make a new feature based on a non-linear combination of other features)

Classifiers

- LinearSVC: https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html#sklearn.svm.LinearSVC
- SVC: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC

SUMMARY

So Many Options

https://github.com/mdaugherity/PatternRecognition2018/blob/master/Tutorial%206.ipynb

Support Vector Machines

- User's Guide - http://scikit-learn.org/stable/modules/svm.html
- API - http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

SVC is the support vector classifier, but there are other classes for a linearsvc and "Nu"svc. The training points closest to the boundary are called the **support vectors** and these are used specially to adjust the margins. SVC training minimizes weight vector plus a constant C times margin erros (Flach eq 7.11 and pgs 216-219, "Soft Margin SVM"). Has the ability to use non-linear **kernels** to make decision boundaries that aren't just straight lines!

Logistic regression

- User's Guide - http://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
- API - http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

Confusingly named, since it does classification and not regression. Very similar to linear SVC. Better at finding probabilites on large datasets than SVC. Less fan but that's not always a bad thing...

Stochastic Gradient Descent

- User's Guide - http://scikit-learn.org/stable/modules/sgd.html
- API - http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html

SGD is efficient for huge datasets, and also the basis for **neural networks**. The method is explain in Flach section 7.2 as the **Perceptron**. (In sklearn, the perceptron class is a special case of SGD). Training occurs by updating the weights for each training point (whereas the other two require the full training set, which is why SGD is far more efficient in speed and memory).

Classifier Comparison

Name	sklearn	Algorithm	Params	Training	Pros	Cons	Notes
Nearest Neighbors (kNN)	neighbors.KNeighborsClassifier	Finds “closest” point in training data	n_neighbors=5 voting weights = (uniform / distance)	Easy! Just copies training data	• Simple to use • Understandable	• SLOW for big datasets	• Always scores 100% on training when k=1 • Increasing k averages over outliers
Decision Tree	tree.DecisionTreeClassifier	20 questions, learns rules (yes/no questions on one feature at a time) that minimizes impurity	max_depth = None	Easy! Default max_depth=None will overfit	• Understandable • Super fast classification	• Overfits by default • “Stair-step” decision boudaries	• Limit max_depth!
Gaussian Naïve Bayes	naive_bayes.GaussianNB	Fits 1D gauss to each feature	None (but see note about priors)	Easy! Just finds mean and var of each feature	• Simple to use • Understandable • Great if your data is gaussian	• Terrible if your problem isn’t gaussian	• Priors learned from y by default, can specify other values
ElasticNetCV (linear with L1 and L2)	linear_model.ElasticNetCV	Sets $y = \pm 1$ and fits a line, uses one-vs-rest for multiclass	L1_ratio =0.5, alpha found automatically	Easy! Automatic CV search takes care of alpha	• Fast and easy • Slopes and L1 reg show feature importance	• Data better be linear!	• good L1_ratios to try [.1, .5, .7, .9, .95, .99, 1] (i.e. don’t get too close to 0)
SVC (Support Vector Machine classifier)	svm.SVC	Fit based only on maximizing margin at border between classes	C=1.0 (inverse of alpha) kernel={‘linear’, ‘poly’, ‘rbf’} degree=3 (for poly only)	Medium – need a grid search	• Fast and stable • Non-linear kernels available	• Only uses points near boundary, can be dominated by outliers or noise	• sklearn has LinearSVC without the fancy options

Classifier	Regularization	Parameters	Notes
RidgeClassifier	L2	Alpha = 1.0	
RidgeClassifierCV	L2	Alpha=(0.1, 1.0, 10.0)	
Lasso	L1	Alpha=1.0	
LassoCV	L1	Alpha found automatically	Automatic search instead of grid
ElasticNet	Both	Alpha=1.0, l1_ratio=0.5	L1_ratio=1 is pure L1 L1_ratio=0 is pure L2
ElasticNetCV	Both	L1_ratio =0.5, alpha found automatically	List of good l1_ratios to try [.1, .5, .7, .9, .95, .99, 1] (i.e. don't get too close to 0)
LogisticRegression	Both	L1_ratio C=1.0	Alpha=1/2C. No reg = Large C
LogisticRegressionCV	Both	Cs defaults to 10 in range [1e-4, 1e4] L1_ratios must specify	