How good is your model?

SUPERVISED LEARNING WITH SCIKIT-LEARN

George Boorman
Core Curriculum Manager, DataCamp

datacamp

Classification metrics

- Measuring model performance with accuracy:
  - Fraction of correctly classified samples
  - Not always a useful metric

Accuracy is not always a useful metric

Class imbalance

- Classification for predicting fraudulent bank transactions
  - 99% of transactions are legitimate; 1% are fraudulent (Assume this)
- Could build a classifier that predicts NONE of the transactions are fraudulent
  - 99% accurate!
  - But terrible at actually predicting fraudulent transactions
  - Fails at its original purpose
- Class imbalance: Uneven frequency of classes
- Need a different way to assess performance

This situation where one class in more frequent is known as class imbalance. Here the class of legitimate transactions has way more instances than the class of fraudulent transactions. This is a common situation in practise.

So we need a different way to assess the performance of a model.

Confusion matrix for assessing classification performance

- Confusion matrix

Given a binary classifier such as a fraudulent transaction example we can create a 2*2 matrix that summarizes performance called a confusion matrix. Given any model we can fill up the confusion matrix given its predictions.

	Predicted: Legitimate	Predicted: Fraudulent
Actual: Legitimate	True Negative	False Positive
Actual: Fraudulent	False Negative	True Positive

Assessing classification performance

	Predicted: Legitimate	Predicted: Fraudulent
Actual: Legitimate	True Negative	False Positive
Actual: Fraudulent	False Negative	True Positive

Assessing classification performance

	Predicted: Legitimate	Predicted: Fraudulent
Actual: Legitimate	True Negative	False Positive
Actual: Fraudulent	False Negative	True Positive

Assessing classification performance

	Predicted: Legitimate	Predicted: Fraudulent
Actual: Legitimate	True Negative	False Positive
Actual: Fraudulent	False Negative	True Positive

Assessing classification performance

	Predicted: Legitimate	Predicted: Fraudulent
Actual: Legitimate	True Negative	False Positive
Actual: Fraudulent	False Negative	True Positive

True positive : Number of fraudulent transactions correctly labeled.

Assessing classification performance

	Predicted: Legitimate	Predicted: Fraudulent
Actual: Legitimate	True Negative	False Positive
Actual: Fraudulent	False Negative	True Positive

True negatives : Number of legitimate transactions that are correctly labeled

Assessing classification performance

	Predicted: Legitimate	Predicted: Fraudulent
Actual: Legitimate	True Negative	False Positive
Actual: Fraudulent	False Negative	True Positive

False negatives : Number of fraudulent transactions incorrectly labeled

Assessing classification performance

	Predicted: Legitimate	Predicted: Fraudulent
Actual: Legitimate	True Negative	False Positive
Actual: Fraudulent	False Negative	True Positive

False positives : Number of legitimate transactions incorrectly labeled

Usually the class of interest is called the positive class. Here the positive class is the illegitimate transactions.

Assessing classification performance

	Predicted: Legitimate	Predicted: Fraudulent
Actual: Legitimate	True Negative	False Positive
Actual: Fraudulent	False Negative	True Positive

- Accuracy: This is the sum of true predictions divided by the total sum of the matrix

$$\frac{tp + tn}{tp + tn + fp + fn}$$

Precision

	Predicted: Legitimate	Predicted: Fraudulent
Actual: Legitimate	True Negative	False Positive
Actual: Fraudulent	False Negative	True Positive

- Precision — Number of true positives divided by the sum of all the positive predictions. It is also known as the positive predictive value.

$$\frac{true\ positives}{true\ positives + false\ positives}$$

- High precision = lower false positive rate
- High precision: Not many legitimate transactions are predicted to be fraudulent

In our case precision is the number of correctly labeled fraudulent transactions divided by the total number of transactions that are classified as fraudulent.

Recall

	Predicted: Legitimate	Predicted: Fraudulent
Actual: Legitimate	True Negative	False Positive
Actual: Fraudulent	False Negative	True Positive

- Recall This is also called sensitivity.

$$\frac{true\ positives}{true\ positives + false\ negatives}$$

- High recall = lower false negative rate
- High recall: Predicted most fraudulent transactions correctly

F1 score

- F1 Score: $2 * \frac{precision * recall}{precision + recall}$

This is the harmonic mean of precision and recall. This metric gives equal weight to precision and recall. Therefore it factors in the number of errors made by the model and type of errors. The F1 score favours models with similar precision and recall and is a useful metric if we are seeking a model which performs reasonably well across both metrics.

Confusion matrix in scikit-learn

```
from sklearn.metrics import classification_report, confusion_matrix
knn = KNeighborsClassifier(n_neighbors=7)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4,
                                                    random_state=42)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
```

Confusion matrix in scikit-learn

```
print(confusion_matrix(y_test, y_pred))
```

```
[[1106   11]
 [ 183   34]]
```

Classification report in scikit-learn

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.86      0.99      0.92      1117
           1       0.76      0.16      0.26       217

    accuracy                           0.85      1334
   macro avg       0.81      0.57      0.59      1334
weighted avg       0.84      0.85      0.81      1334
```

0.76 and 0.16 for the churn class highlights how poor the models recall is for the churn class. Support represents the number of instances for each class within the true labels.

Let's practice!

SUPERVISED LEARNING WITH SCIKIT-LEARN

Logistic regression and the ROC curve

SUPERVISED LEARNING WITH SCIKIT-LEARN

George Boorman
Core Curriculum Manager, DataCamp

datacamp

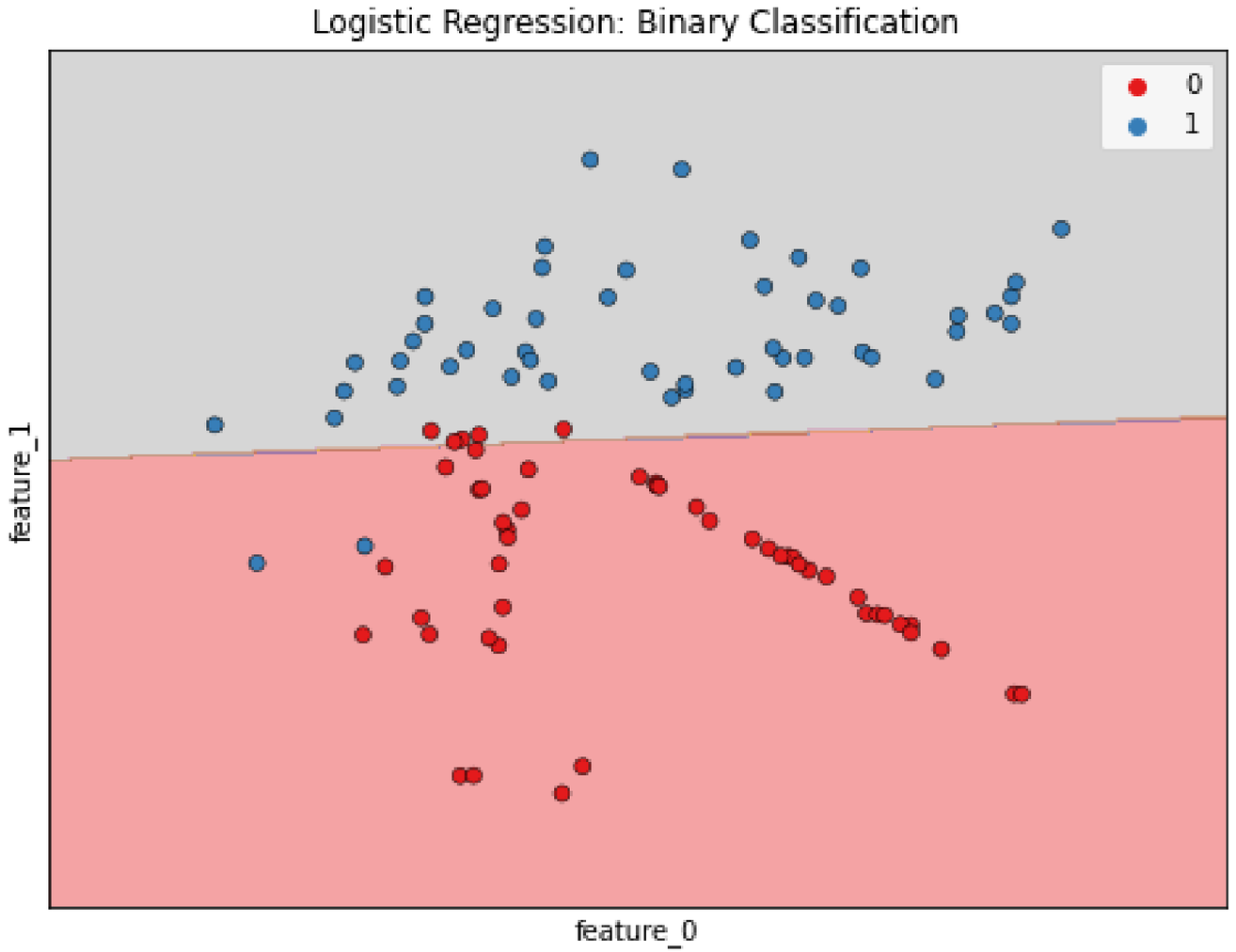
Logistic regression for binary classification

- Logistic regression is used for classification problems
- Logistic regression outputs probabilities
- If the probability, $p > 0.5$:
  - The data is labeled `1`
- If the probability, $p < 0.5$:
  - The data is labeled `0`

This model calculates the probability p that an observation belongs to a binary class.
Using the diabetes dataset as an example, if p > or equal to 0.5 we label the data is 1 representing a prediction that an individual is more likely to have diabetes.
If p < 0.5 we label it as 0 to represent that they are more likely to not have diabetes.

Linear decision boundary

Logistic regression produces a linear decision boundary as we can see in this image.



Logistic regression in scikit-learn

```
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression()
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
                                                    random_state=42)
logreg.fit(X_train, y_train)
y_pred = logreg.predict(X_test)
```

Instantiation of the classifier

In this video we use the churn dataset

Predicting probabilities

```
y_pred_probs = logreg.predict_proba(X_test)[:, 1]
print(y_pred_probs[0])
```

```
[0.08961376]
```

We can predict the probabilities of each instance belonging to a class by calling logistic regression's predict_proba method and passing the test features. This returns a 2 dimensional array with probabilites for both classes. In this case that the individual did not churn or did churn respectively. We slice the second column representing the positive class probabilities and store the results as y_pred_probs.
Here we see that the model predicts a probability of 0.089 that the first observation has churned.

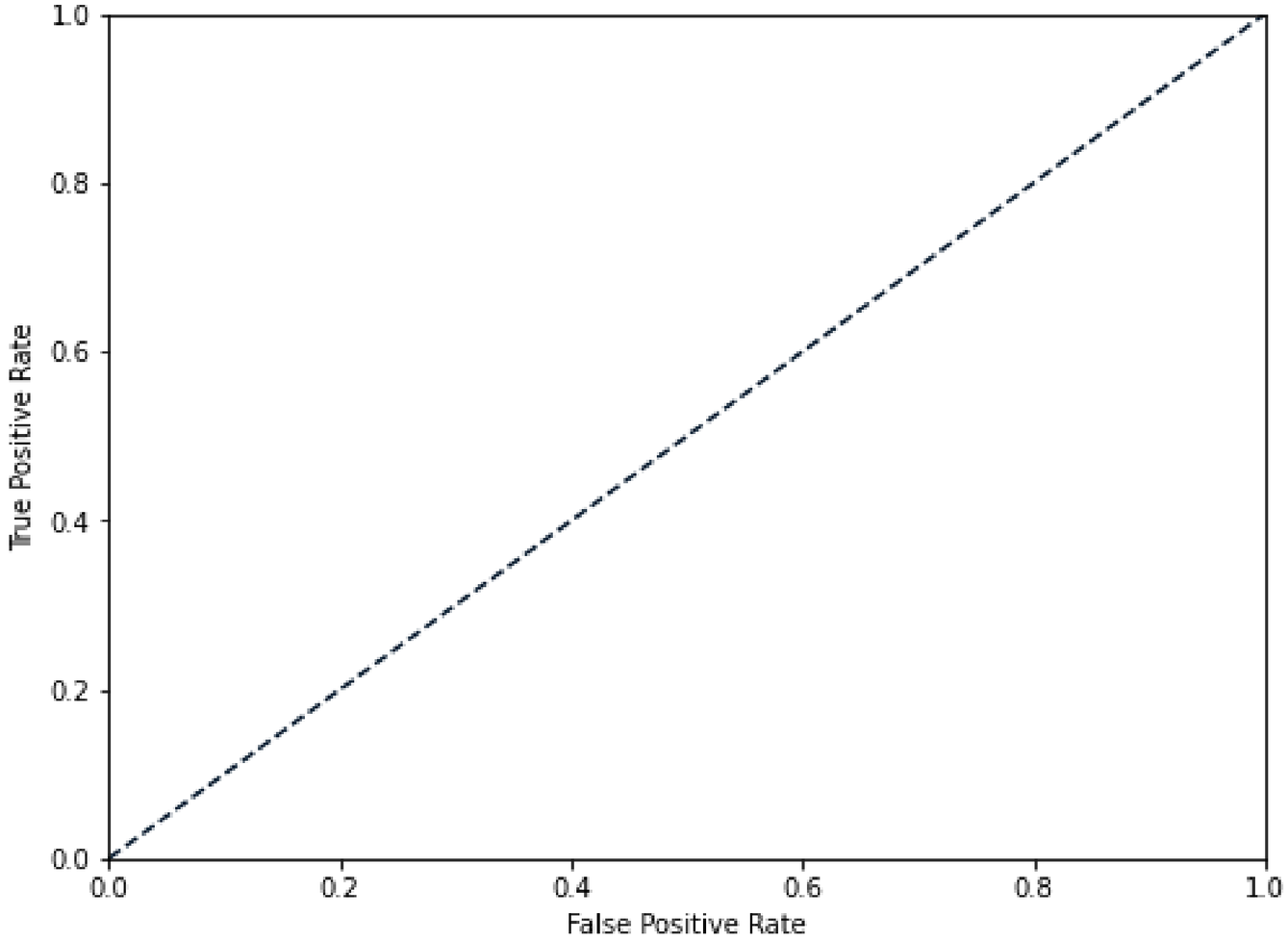
Probability thresholds

The default probability threshold for logistic regression in scikit-learn is 0.5. This threshold can also be applied to other models such as KNN.

- By default, logistic regression threshold = 0.5
- Not specific to logistic regression
  - KNN classifiers also have thresholds
- What happens if we vary the threshold?

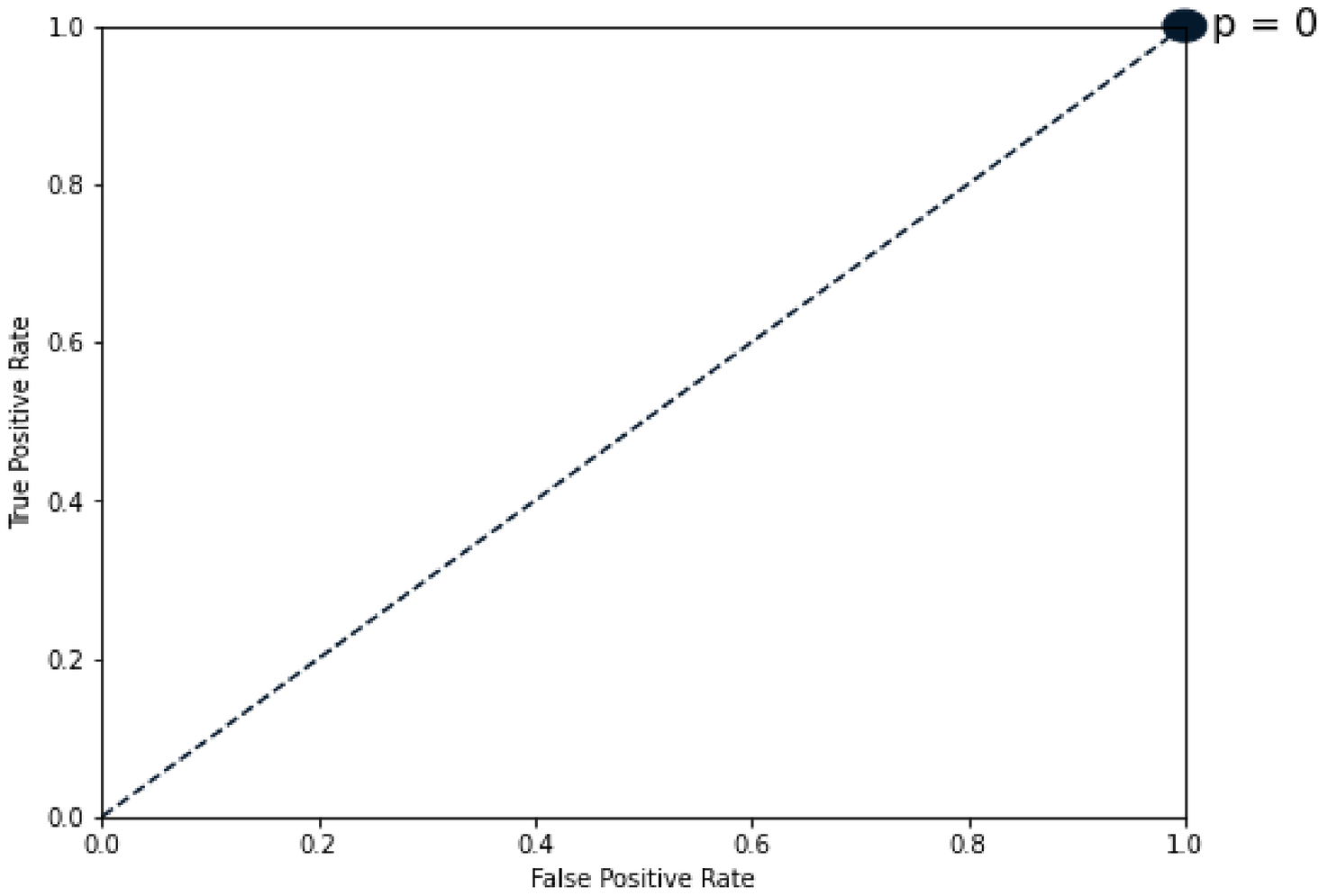
The ROC curve

We can use a Reciever Operating Characteristic or ROC curve how diffferent thresholds affect the true positive and false positive rates. Here the dotted line represents the chance model which randomly guesses labels.

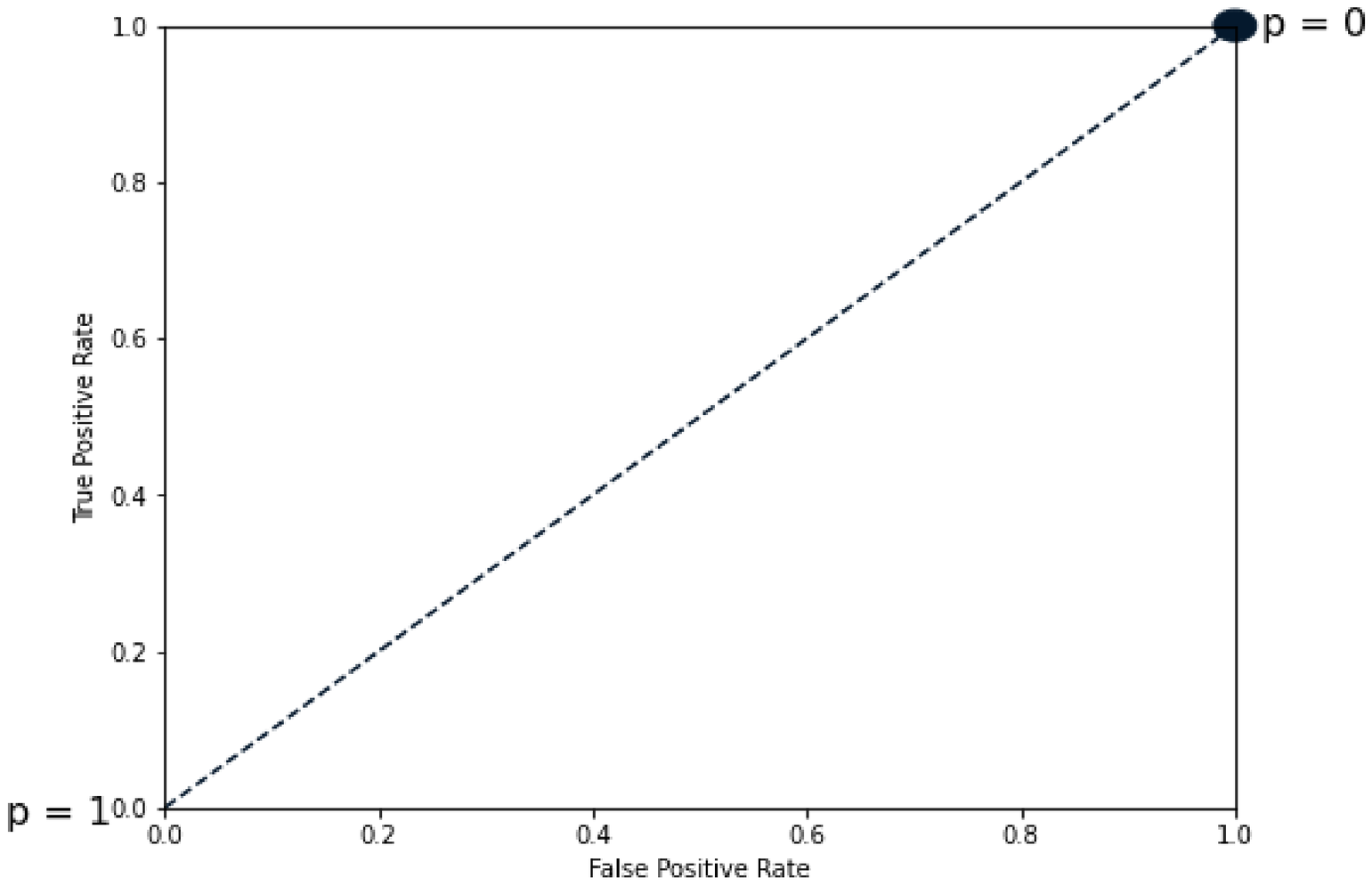


The ROC curve

When the threshold = 0, the model predicts 1 for all observations meanly it will correctly predict all positive values and incorreclty predict all negative values.

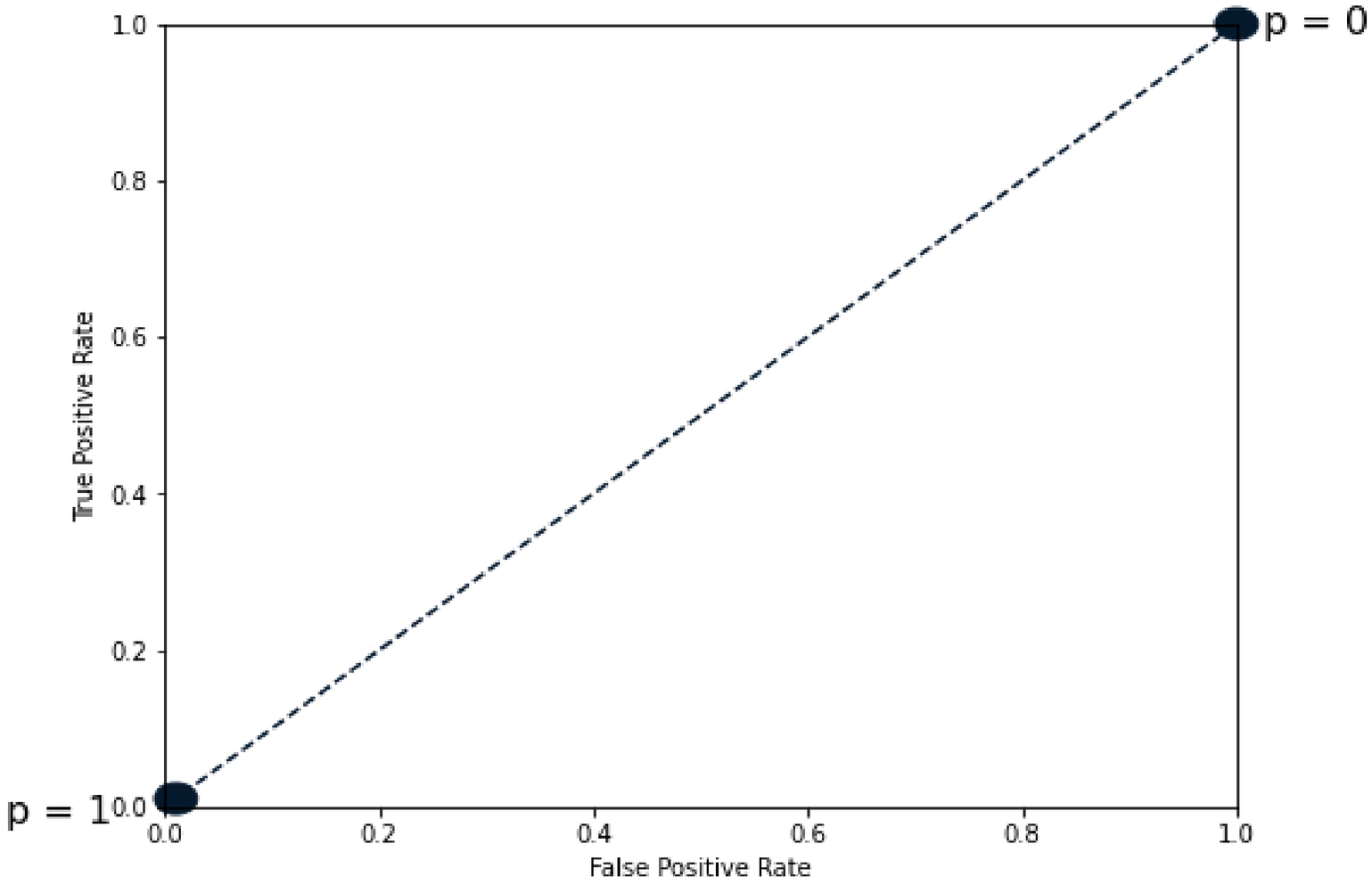


The ROC curve



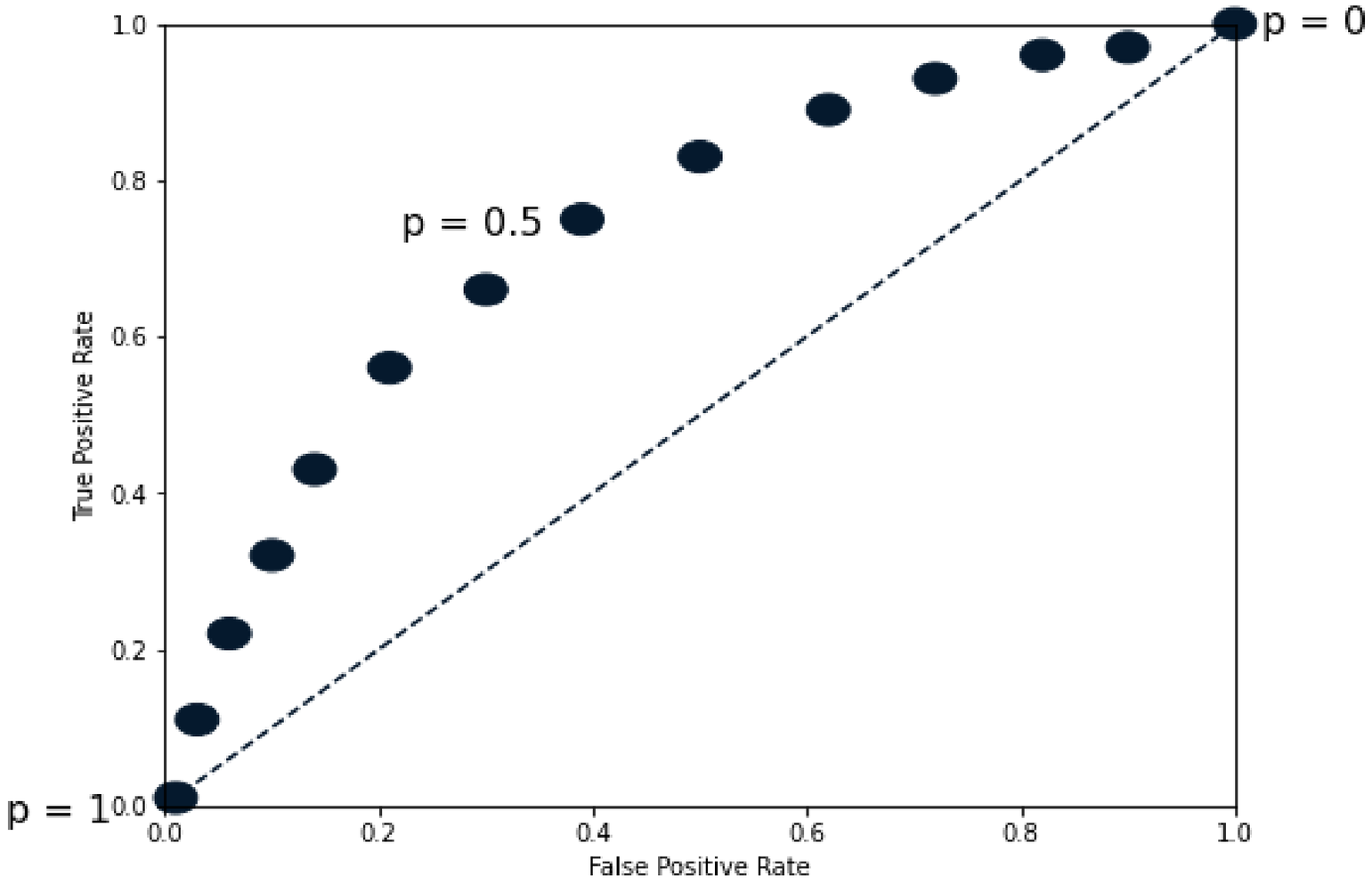
The ROC curve

If the threshold = 1 the model predicts 0 for all data which means that all the true and false positive rates are zero.



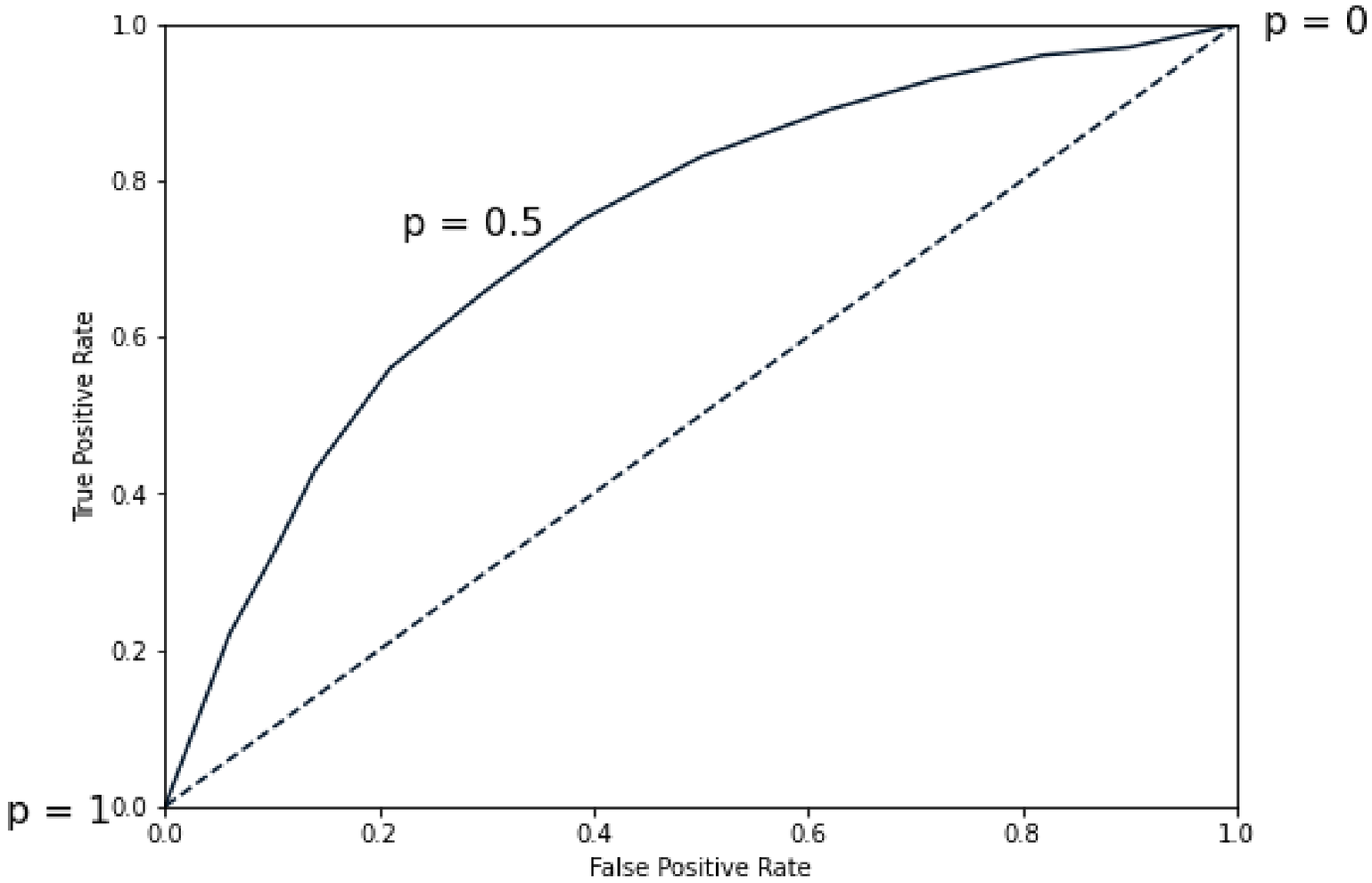
The ROC curve

If we vary the threshold, we get a series of different false positive and true positive rates.



The ROC curve

A line plot of the thresholds helps to visualize the trend.



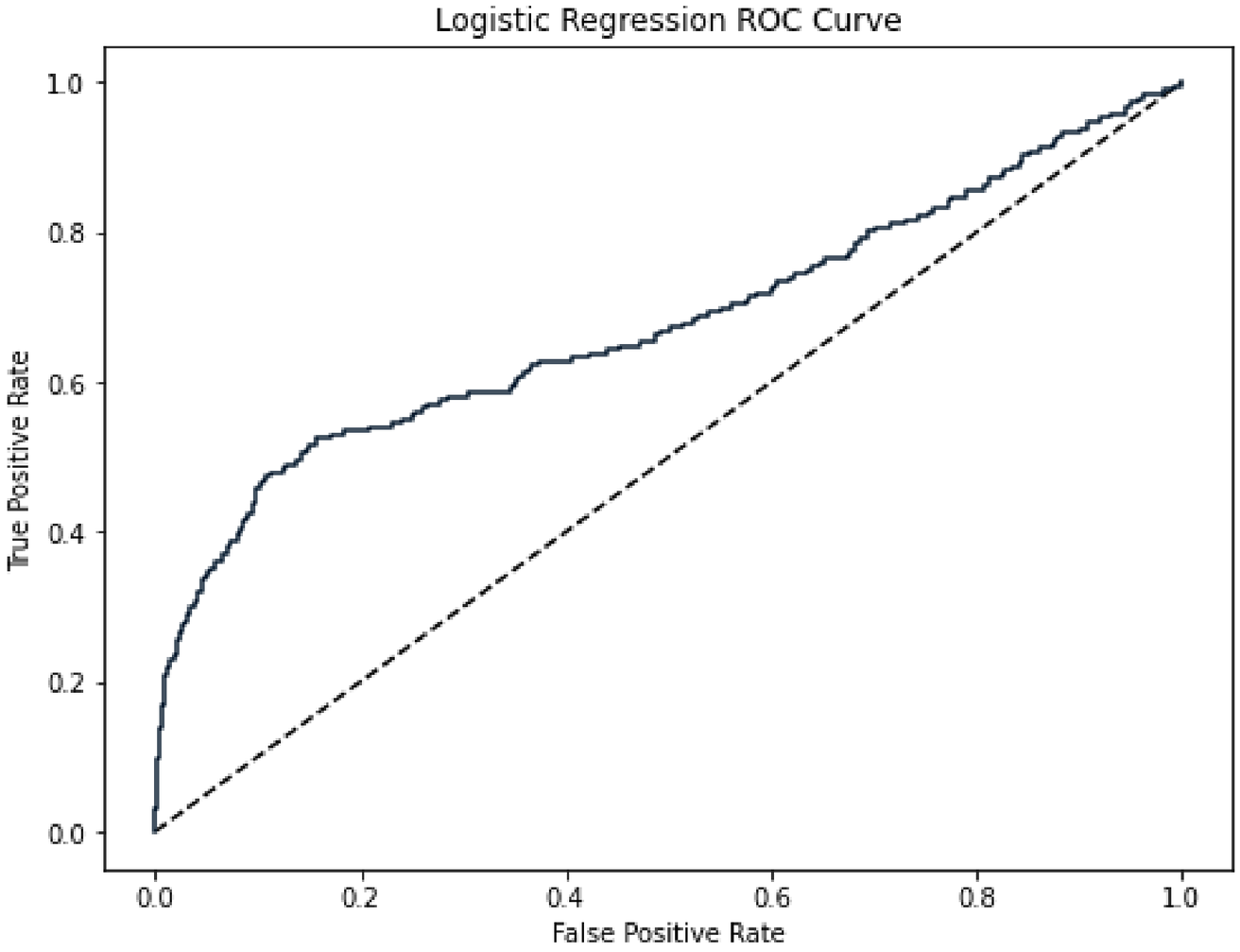
Plotting the ROC curve

```
from sklearn.metrics import roc_curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred_probs)
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr, tpr)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Logistic Regression ROC Curve')
plt.show()
```

To the roc_curve function and pass the test labels as the first argument and the predicted probabilities as the second argument. We unpack the results into 3 variables. False positive rates, true positive rates and the thresholds. We can then plot a dotted line from zero to one along with the fpr and the tpr.

Plotting the ROC curve

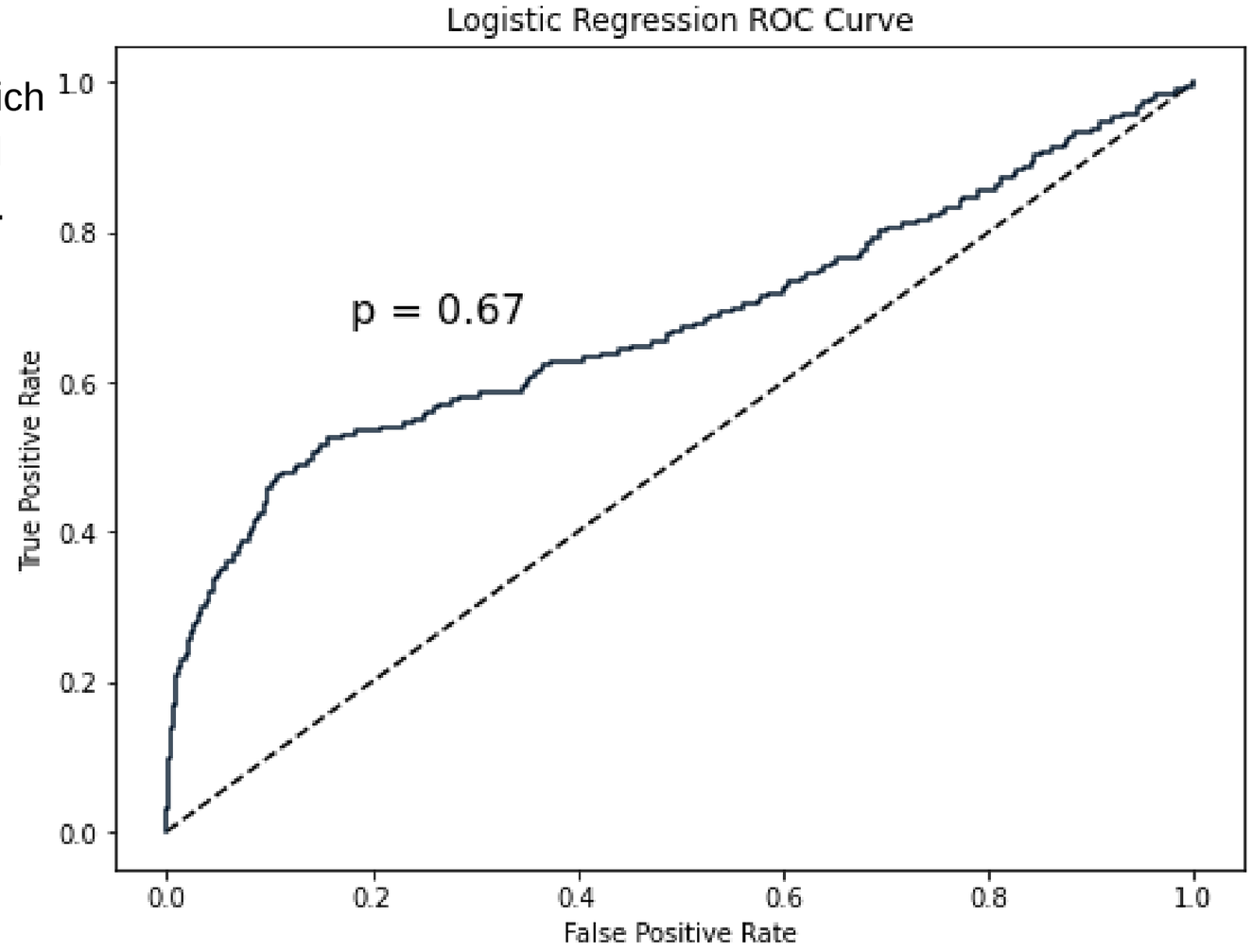
We get a figure such as this. How do we quantify the model's performance based on this plot.



ROC AUC

If we have a model with 1 for true positive rate and 0 for false positive rates this would be the perfect model. Therefore we calculate the area under the ROC curve. A metric known as AUC. Scores range from 0 to 1. With 1 being ideal.

Here the model scores 0.67 which is only 34% better than a model that is making random guesses.



ROC AUC in scikit-learn

```
from sklearn.metrics import roc_auc_score
print(roc_auc_score(y_test, y_pred_probs))
```

```
0.6700964152663693
```

Let's practice!

SUPERVISED LEARNING WITH SCIKIT-LEARN

Hyperparameter tuning

SUPERVISED LEARNING WITH SCIKIT-LEARN

George Boorman
Core Curriculum Manager

As we know how to evaluate our model performance let's understand how to optimize our model.

Hyperparameter tuning

- Ridge/lasso regression: Choosing `alpha`
- KNN: Choosing `n_neighbors`
- Hyperparameters: Parameters we specify before fitting the model
  - Like `alpha` and `n_neighbors`

Recall that we had to choose a value for alpha in ridge and lasso regression before fitting it. Likewise before fitting and predicting KNN we choose n_neighbors. Parameters that we specify before fitting a model like alpha and n_neighbors are called hyperparameters.

A fundamental step in building a successful model is choosing the correct hyperparameters.

Choosing the correct hyperparameters

1. Try lots of different hyperparameter values
2. Fit all of them separately
3. See how well they perform
4. Choose the best performing values

When fitting different hyperparmeter values we use cross-validation to avoid overfitting the hyperparameters to the test set.

- This is called **hyperparameter tuning**
- It is essential to use cross-validation to avoid overfitting to the test set
- We can still split the data and perform cross-validation on the training set
- We withhold the test set for final evaluation

We withhold the test set and use it for evaluating the tuned model.

Grid search cross-validation

One approach for hyper parameter tuning is called grid search where we choose a grid of hyperparameter values to try.

	11		
	8		
	5		
n_neighbors	2		
		euclidean	manhattan
		metric	

For example we can choose across 2 hyper parameters for a KNN model, the type of metric and a different number of labels. Here we have nighbors from 2 to 11 and increments in 3. And we have two labels euclidean and manhattan.

Therefore we can make a grid of values as shown in the next slide.

Grid search cross-validation

	11	0.8716	0.8692
	8	0.8704	0.8688
	5	0.8748	0.8714
n_neighbors	2	0.8634	0.8646
		euclidean	manhattan
		metric	

We perform k fold cross validation for each combination of hyperparameters. The mean scores for each combination is stored here.

Grid search cross-validation

	11	0.8716	0.8692
	8	0.8704	0.8688
	5	0.8748	0.8714
n_neighbors	2	0.8634	0.8646
		euclidean	manhattan
		metric	

We then choose the hyperparameters that perform the best as shown here.

Lets perform a grid search on a regression model using our sales dataset.

GridSearchCV in scikit-learn

First we import GridSearchCV. After that we instantiate KFold. We then specify the names and values of the hyperparameters that we wish to tune as the keys and values in a dictionary param_grid.

```
from sklearn.model_selection import GridSearchCV
kf = KFold(n_splits=5, shuffle=True, random_state=42)
param_grid = {"alpha": np.arange(0.0001, 1, 10),
              "solver": ["sag", "lsqr"]}
ridge = Ridge()
ridge_cv = GridSearchCV(ridge, param_grid, cv=kf)
ridge_cv.fit(X_train, y_train)
print(ridge_cv.best_params_, ridge_cv.best_score_)
```

Here we instantiate our model. We then call GridSeachCV and pass our model, the grid we wish to tune over and set cv=kf. This returns a gridsearch object that we can then fit to the training data.

This fit performs the actual cross validated gridseach.

```
{'alpha': 0.0001, 'solver': 'sag'}
0.7529912278705785
```

We can then print the model's attributes best_params_ and best_score_ respectively to retrieve the hyperparameters that performs the best along with the mean cross validation score over that fold.

Limitations and an alternative approach

- 3-fold cross-validation, 1 hyperparameter, 10 total values = 30 fits
- 10 fold cross-validation, 3 hyperparameters, 30 total values = 900 fits

Grid seach is great but the
number of fits = no.of hyperparameters * no. of values * no. of folds.

Therefore it doesn't scale well. So performing 3 fold cross validation for one hyperparameter for 10 values each means 30 fits. While 10 fold cross validation on 3 hyperparameters of 10 values each equals 900 fits.

However there is another way. We can follow a random search which picks random hyperparameter values rather than exhaustively searching for all options.

RandomizedSearchCV

In the line 5 we call the RandomizedSearchCV and using the same arguments and variables as before but we can optionally set the n_iter argument which determines the number of hyperparameter values tested.

```
from sklearn.model_selection import RandomizedSearchCV
kf = KFold(n_splits=5, shuffle=True, random_state=42)
param_grid = {'alpha': np.arange(0.0001, 1, 10),
              "solver": ['sag', 'lsqr']}
ridge = Ridge()
ridge_cv = RandomizedSearchCV(ridge, param_grid, cv=kf, n_iter=2)
ridge_cv.fit(X_train, y_train)
print(ridge_cv.best_params_, ridge_cv.best_score_)
```

So 5 fold cross validation with n_iter set to 2 performs 10 fits. Again we can access the best hyperparameters and best score. In this case

```
{'solver': 'sag', 'alpha': 0.0001}
0.7529912278705785
```

Evaluating on the test set

```
test_score = ridge_cv.score(X_test, y_test)
print(test_score)
```

```
0.7564731534089224
```

We can evaluate the model performanve on the test set by passing it to the .score method.

Actually it performs slightly better than the best score in our grid seach.

Let's practice!

SUPERVISED LEARNING WITH SCIKIT-LEARN